Глава 15

Тестирование

В этой главе

- Введение
- Новые возможности блочного тестирования в Microsoft Dynamics AX 2012
- Инструменты тестирования Microsoft Visual Studio 2010
- Собираем все элементы вместе

Введение

Обеспечение качественного пользовательского взаимодействия для ERP-системы, такой как Microsoft Dynamics AX 2012, может быть непростой задачей. Даже в стандартной поставке от Microsoft система огромна и сложна, а в типичном внедрении для поддержки специфических задач обычно используется еще один или несколько продуктов от сторонних разработчиков. Наконец, клиент или компания-партнер модифицируют Microsoft Dynamics AX, чтобы еще более повысить уровень ее соответствия бизнес-процессам и политикам компании.

Конечный пользователь не видит или, если на то пошло, не беспокоится о сложностях реализации продукта или о том, кто именно какую функциональность реализовал. Конечный пользователь хочет получить продукт, который позволит ему эффективно, результативно и удобно выполнять ежедневные задачи. Вся экосистема — Microsoft, независимые разработчики ПО, партнеры, внедряющие Microsoft Dynamics AX, и разработчики самого клиента — это часть критически важной цепочки обеспечения качества, на которую полагаются конечные пользователи, когда для достижения своих целей используют Microsoft Dynamics AX 2012.

В этой главе основное внимание уделено новым возможностям и инструментам, помогающим улучшить тестирование сторонних решений на базе Microsoft Dynamics AX 2012, а также модификаций системы. В этой версии реализовано несколько новых возможностей, которые вместе поз-

воляют сделать большой шаг вперед в деле эффективного тестирования решений на базе Microsoft Dynamics AX 2012.

Новые возможности блочного тестирования в Microsoft Dynamics AX 2012

Инфраструктура SysTest для блочного тестирования является частью среды разработки MorphX уже на протяжении нескольких выпусков Microsoft Dynamics AX. Эта инфраструктура может быть очень полезной для традиционного блочного тестирования классов и методов X++. Она также может использоваться для интеграционного тестирования бизнес-логики, реализованной в нескольких классах.

Основы использования инфраструктуры SysTest хорошо изложены в прежних изданиях этой книги и в актуальной документации MSDN. За счет использования атрибутов, появившихся в языке X++, инфраструктура SysTest в Microsoft Dynamics AX 2012 обрела новые возможности, которые вы можете использовать для более гибкой разработки и выполнения ваших тестов. Эти новые возможности описаны в этой главе.

Использование предопределенных атрибутов для тестирования

Атрибуты X++ являются новой возможностью Microsoft Dynamics AX 2012 и описаны в главе 4. В этом разделе описывается то, как вы можете использовать предопределенные атрибуты для тестирования в инфраструктуре SysTest, чтобы сделать разработку и выполнение тестов более гибкими. Как часть инфраструктуры SysTest предоставляются пять атрибутов, которые описаны в табл. 15-1.

Табл. 15–1. Предопределенные атрибуты SysTest

Атрибут для тестирования	Описание	Где применять атрибут
SysTestMethodAttribute	Указывает, что метод является блочным тестом	Применяйте к методам

Табл. 15–1. Предопределенные атрибуты SysTest (*окончание*)

Атрибут для тестирования	Описание	Где применять атрибут
SysTestCheckInTestAttribute	Указывает, что это check-in тест, то есть что он должен выполняться при возврате модифицированных объектов в систему контроля версий, чтобы убедиться в должном уровне качества модификации	Применяйте к методам или классам
SysTestNonCheckInTestAttribute	Указывает, что это – не check-in тест	Применяйте к методам
SysTestTargetAttribute (<имя>, <тип>)	Указывает на объект приложения, который тестируется данным тестом; например, это может быть класс, таблица или форма. Этот атрибут принимает два параметра: ■ *Имя* – строка с названием тестируемого элемента приложения; ■ *Тип* – тип тестируемого элемента приложения	Применяйте к классам
SysTestInactiveTestAttribute	Указывает, что класс или метод не используется	Применяйте к методам

В предыдущих версиях инфраструктуры SysTest для указания назначения класса или метода широко использовались соглашения по именованию. Чтобы обозначить, что класс используется для сбора тестового покрытия, применялся шаблон имени *<ЦелевойКласс>Test*. (В качестве альтернативы использованию этого соглашения по именованию вы также могли переопределить метод *testsElementName* вашего тестового класса.) Названия всех методов, предназначенных для тестирования, в классе-наследнике *SysTestCase* должны были начинаться со строки *test*. За счет использования *SysTestTargetAttribute* и *SysTestMethodAttribute* вы можете более явно выражать назначение вашего кода блочных тестов. В следующем примере показывается, как вы можете использовать эти атрибуты.

```
[SysTestTargetAttribute(classStr(Triangles), UtilElementType::Class)]
public class TrianglesTest extends SysTestCase
{
}
```

```
[SysTestMethodAttribute]
public void testEQUILATERAL()
{
    Triangles triangle = new Triangles();
    this.assertEquals(TriangleType::EQUILATERAL, triangle.IsTriangle(10, 10, 10));
}
```

Новые предопределенные атрибуты дают возможность создавать фильтры, чтобы вы могли выполнять определенные тесты из тех, что показаны на рис. 15-1. Для этих целей можно использовать *SysTestCheckInTestAttribute*, *SysTestNonCheckInTestAttribute* и *SysTestInactiveTestAttribute*. На форме параметров, доступной на панели инструментов блочного тестирования, вы можете выбрать фильтр, который будет использоваться при выполнении тестов. Более подробную информацию о создании фильтра вы найдете в следующем разделе.

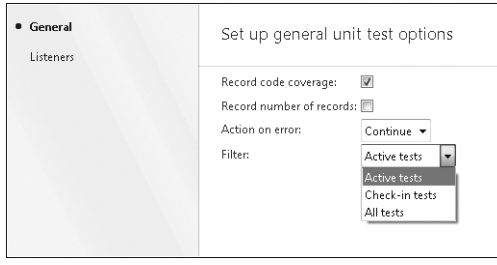

Рис. 15–1. Список фильтрации на форме Параметры

Прекрасным способом поддерживать высокий уровень качества вашего кода является выполнение регрессионных тестов перед регистрацией кода в системе контроля версий. Но не всегда вам может хотеться выполнять все тесты, потому что длительность их выполнения может стать проблемой. Здесь-то и пригодятся атрибуты *SysTestCheckInTestAttribute* и *SysTestNonCheckInTestAttribute*. Более подробную информацию о системах контроля версий вы можете найти в главе 2 .

Для указания того, какие тесты нужно выполнять при регистрации кода в системе контроля версий (т.н. check-in), выполните следующие шаги.

1. Наградите методы тестирования атрибутом *SysTestCheckInTestAttribute* или *SysTestNonCheckInTestAttribute*. Заметьте, что по умолчанию тест *не* является check-in тестом, поэтому, чтобы сделать его таковым, вам нужно указать *SysTestCheckInTestAttribute*. Наилучшим подходом явля-

ется явное указание одного из двух атрибутов для всех тестов, как в следующем примере.

```
[SysTestMethodAttribute, SysTestCheckInTestAttribute]
public void testEQUILATERAL()
{
    Triangles triangle = new Triangles();
    this.assertEquals(TriangleType::EQUILATERAL, triangle.IsTriangle(10, 10, 10));
}
```

2. Создайте новый проект тестирования и поместите все классы блочных тестов с check-in тестами в этот проект.
3. На форме Настройки для созданного проекта тестирования (щелкните правой кнопкой мыши по названию проекта и выберите Настройки) укажите в качестве фильтра Проверки при возврате (Check-in Tests).
4. В меню Контроль версий > Параметры системы выберите ваш проект в списке проектов тестирования.

При следующем возврате измененных объектов приложения в систему контроля версий будут выполнены соответствующие тесты, и сообщения о результатах их работы выведены в Infolog.

Создание атрибутов для тестирования и фильтров

Предопределенные атрибуты для тестирования, описанные в предыдущем разделе, являются хорошей отправной точкой в деле организации ваших тестов и того, чтобы более явно обозначать, какие из них для чего нужны. Хорошо организованная стратегия использования проектов блочного тестирования также не помешает. Однако велика вероятность, что для своих проектов разработки вы захотите продвинуться в организации тестов еще дальше. К счастью, можно расширить возможности атрибутов для тестирования, создав собственные атрибуты и фильтры.

Как указывалось ранее в этой главе, инфраструктура SysTest может быть полезной как для тестирования отдельных классов, так и для интеграционного тестирования бизнес-логики, реализованной в нескольких классах. Однако в определенных сценариях может быть полезно выполнять только интеграционные тесты, потому что они больше ориентированы на функциональность. К примеру, вы могли бы захотеть запускать только интеграционные тесты, когда переносите модификации из тестовой среды в среду для подготовки к переносу на рабочее приложение. В этом разделе показывается, как вы можете создать новый атрибут и соот-

ветствующий фильтр для использования в интеграционных тестах. Сначала создайте новый атрибут – это достаточно прямолинейный процесс, потому что нужно лишь создать класс-наследник *SysTestFilterAttribute*.

```
class SysTestIntegrationTestAttribute extends SysTestFilterAttribute
{
}
```

Теперь вы можете использовать этот атрибут для нового тестового метода, как показано ниже.

```
[SysTestMethodAttribute, SysTestIntegrationTestAttribute]
public void testIntegratedBusinessLogic()
{
    this.assertFalse(true);
}
```

Но хотя атрибут достаточно информативен для любого, кто будет читать код этого теста, он не особо полезен для выполнения тестов, пока вы не добавите возможность фильтрации по нему в выпадающем списке фильтров. Для этого необходимо реализовать *тестовую стратегию* для *IntegrationTestAttribute*. Термин «стратегия» использован потому, что тестовая стратегия реализуется с использование шаблона проектирования Стратегия.

Для начала добавьте в перечисление *SysTestFilterStrategyType* дополнительный элемент, как показано на рис. 15-2. Не забудьте указать для нового элемента соответствующую метку.

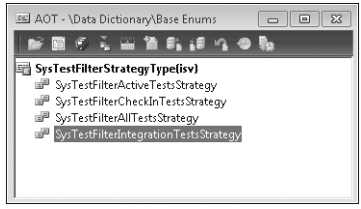

Рис. 15–2. Добавление элемента в перечисление *SysTestFilterStrategyType*

Затем реализуйте стратегию в классе с тем же именем, как у элемента перечисления. Этот класс наследует от *SysTestFilterStrategy* и объявлен следующим образом:

```
class SysTestFilterIntegrationTestsStrategy extends SysTestFilterStrategy
{
}
```

Наиболее простой способ реализации этой стратегии – следовать шаблону одного из имеющихся классов *SysTestFilter<атрибут>TestsStrategy*. В этом случае вам нужно реализовать лишь два метода. Метод *construct* возвращает новый экземпляр класса – вы вскоре им воспользуетесь.

```
public static SysTestFilterIntegrationTestsStrategy construct()
{
    return new SysTestFilterIntegrationTestsStrategy();
}
```

Основная работа класса выполняется в методе *isValid*. Он определяет, надо ли включать метод теста в список выполняемых тестов. Вот его реализация для *SysTestFilterIntegrationTestsStrategy*:

```
public boolean isValid(classId _classId, identifierName _method)
{
    SysDictMethod method;
    DictClass dictClass;

    method = this.getMethod(_classId, _method);
    if (method)
    {

        //
        // Если у тестового метода есть атрибут интеграционного теста, то используем его
        //
        if (method.getAttribute(attributestr(SysTestIntegrationTestAttribute)))
        {
            return true;
        }
    }

    //
    // Если у класса теста есть атрибут интеграционного теста, то используем его
    //
    dictClass = new DictClass(_classId);
    if (dictClass.getAttribute(attributestr(SysTestIntegrationTestAttribute)))
    {
        return true;
    }
    return false;
}
```

Примечание. Для реализации поведения, как у атрибута *SysTestInactiveTestAttribute*, который также может применяться к тестовым методам, нужно реализовать дополнительную логику. Соответствующий код был опущен для упрощения примера.

Чтобы новый атрибут для интеграционного тестирования заработал, остался последний штрих. Метод *newType* класса *SysTestFilterStrategy* создает соответствующий тип на основе выбора из списка фильтрации, и в этот метод нужно добавить дополнительный блок *case*, как показано в следующем примере.

```
public static SysTestFilterStrategy newType(SysTestFilterStrategyType _type)
{
    SysTestFilterStrategy strategy;

    switch (_type)
    {
                <несущественный код удален>
        // создание стратегии интеграционного тестирования
        case SysTestFilterStrategyType::SysTestFilterIntegrationTestsStrategy:
            strategy = SysTestFilterIntegrationTestsStrategy::construct();
            break;
            default:
                throw error(error::wrongUseOfFunction(funcname()));
    }

    strategy.parmFilterType(_type);
    return strategy;
}
```

Теперь в списке фильтрации доступен пункт Интеграционный тест (рис. 15-3), и при его выборе запускаются только те методы тестирования, у которых есть атрибут *SysTestIntegrationTestAttribute*.

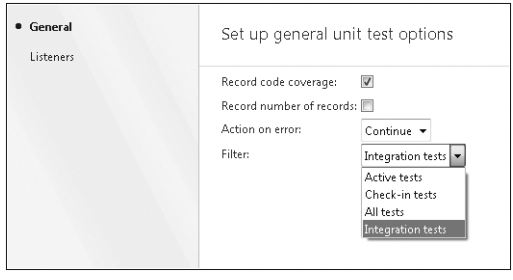

Рис. 15–3. Список фильтрации с добавленным вариантом выбора

Инструменты тестирования Microsoft Visual Studio 2010

Возможности блочного тестирования SysTest чрезвычайно важны для разработчиков, тестирующих Microsoft Dynamics AX, но большая часть тестирования выполняется не разработчиками. Функциональное тестирование, проверка соответствия продукта требованиям клиента, как правило, выполняются кем-либо в должности консультанта, бизнес-аналитика или, возможно, кем-то, чья основная работа связана с использованием Microsoft Dynamics AX для решения каждодневных задач. У этих функциональных тестировщиков есть кое-что общее.

- Они являются экспертами по продукту и его использованию для решения бизнес-задач.
- Они не обучались на программистов или тестировщиков ПО. Необходимость написания нетривиального кода, если таковая возникнет при тестировании, будет для них серьезным препятствием.
- Хотя они не обучались на тестировщиков ПО, они тем не менее достаточно хорошо умеют выполнять тестирование. Их любознательная натура делает их мастерами в поиске проблемных мест в продукте.
- Для выполнения повторяющихся тестов они очень хотели бы иметь поддержку автоматизированного тестирования, но они также верят, что исследовательский подход с использованием ручного выполнения тестов – это лучший способ проверки системы и поиска критических ошибок.

В редакции Microsoft Visual Studio 2010 Ultimate и Visual Studio Test Professional входит Microsoft Test Manager, приложение, которое было создано специально для такого типа тестировщиков. Этот пакет инструментов тестирования хорошо подходит для проектов Microsoft Dynamics AX. Team Foundation Server, необходимый для работы Microsoft Test Manager, также привносит несколько возможностей, нацеленных на контроль качества. Для фазы разработки проекта он предоставляет решение по управлению жизненным циклом приложения (application lifecycle management, ALM) за счет интеграции управления требованиями, управления проектом, системы контроля версий, системы отслеживания ошибок, процессов сборки и инструментов тестирования. Более подробную информацию об ALM вы можете найти в статье «What Is Application Lifecycle Management?» по адре-

су: *http://www.microsoft.com/global/applicationplatform/en/us/RenderingAssets/Whitepapers/What%20is%20Application%20Lifecycle%20Management.pdf.*

В этом разделе основное внимание уделяется рекомендациям по использованию Microsoft Test Manager для проектов Microsoft Dynamics AX. Более подробную информацию об использовании Microsoft Test Manager вы можете найти в разделе MSDN «Краткое руководство по ручному тестированию с использованием Microsoft Test Manager» по адресу: *http://msdn.microsoft.com/ru-ru/library/dd380763.aspx*.

Используйте все аспекты ALM-решения

План контроля качества вашего проекта не должен фокусироваться только на тестировании. Напротив, многие другие факторы могут иметь большее влияние на качество проекта, чем объем проведенного тестирования. Один из ключевых факторов – это *управление конфигурациями*. С помощью ALM-решения Visual Studio вы можете отслеживать все существенные артефакты в вашем процессе разработки ПО. А за счет использования этого ALM-решения на протяжении всего проекта вы можете существенно повысить качество ваших проектов.

Рис. 15-4 описывает законченный цикл разработки, в котором задействован Саймон, консультант по функционалу системы, и Айзек, программист. Как вы можете видеть, процесс включает отслеживание требований, тестовых сценариев, исходного кода, сборок и ошибок. Многие из этих элементов отслеживаются в Team Foundation Server (TFS). Этим также объясняется возможность отследить взаимосвязи между этими артефактами. Для улучшения управления проектом вы также можете включить в этот процесс рабочие задания.

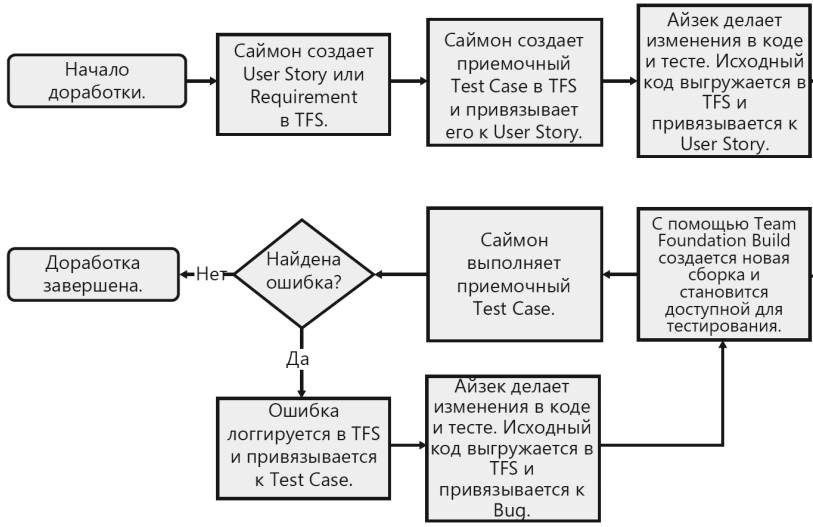

Рис. 15–4. Цикл разработки функционального блока

Используйте подход к разработке на основе приемочного тестирования

В быстро меняющейся обстановке, какой является большинство проектов внедрения Microsoft Dynamics AX, часто наилучшие результаты показывают гибкие методологии разработки. Одна из практик гибких методологий разработки, которая особенно полезна для обеспечения должного уровня качества, – это разработка на основе приемочного тестирования (acceptance test driven development, ATDD). ATDD включает в качестве одного из требований определение критических тестовых сценариев – приемочных тестов – до начала разработки. (Термин «*требование*» тут использован в общем смысле. Требованием может быть пользовательская история, функциональная возможность или другой артефакт, который описывает функциональность, представляющую ценность для клиента.) Контрольные примеры в приемочных тестах проверяют основные несоответствия требованию. Для всестороннего тестирования соответствия реализации требованию нужны дополнительные контрольные примеры. Приемочные тесты должны быть плодом совместной работы разработчика, тестировщика и автора требования. Существенным выигрышем от такой совместной работы является ясность, потому что у вовлеченных в нее людей за-

частую есть свои несформулированные версии того, как, по их мнению, требование должно быть реализовано.

Хотя в формулировке требования не должно быть деталей реализации, для приемочных тестов такие детали необходимы – в объеме, достаточном, чтобы от тестов была польза, но не более того. На рис. 15-5 показан приемочный тест для функции, описанной в предыдущем разделе, – выполнения блочных тестов при возврате модифицированного кода в систему контроля версий. В тестовом примере указаны детали, например формы, которые предположительно будут использованы, но в нем нет указания конкретных названий полей ввода.

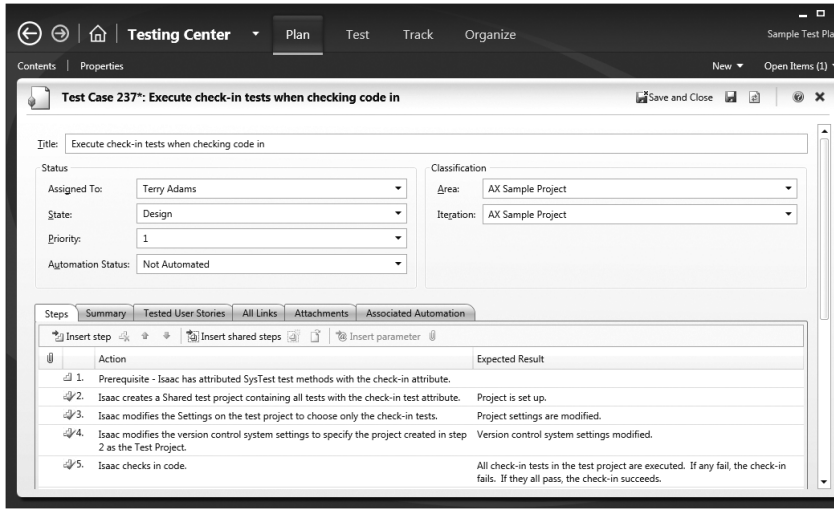

Рис. 15–5. Приемочный тест

После создания контрольного примера свяжите его с требованием на закладке Tested User Stories (тестируемые пользовательские истории; в данном примере пользовательская история является требованием). После установления такой связи форма **Add Link (добавить связь)** будет выглядеть, как на рис. 15-6.

За счет связывания контрольного примера с требованием вы можете воспользоваться приятной функцией **Microsoft Test Manager – построением планов тестирования** на основе пакетов требований и связанных с ними контрольных примеров. Выбирая требование в области плана тестирования Microsoft Test Manager с помощью кнопки Add Requirements,

вы подтяните все контрольные примеры, которые связаны с этим требованием.

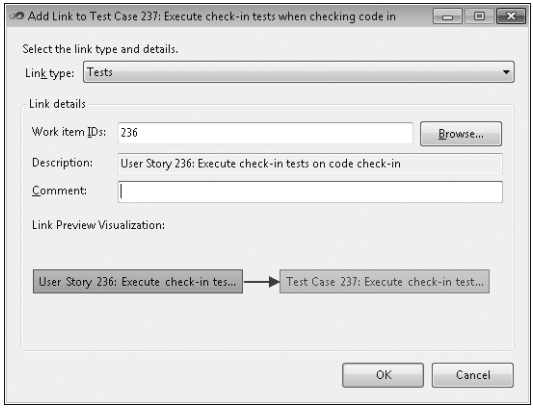

Рис. 15–6. Требование, связанное с контрольным примером

Используйте разделяемые этапы теста

С помощью разделяемых этапов теста вы можете повторно использовать одни и те же этапы в нескольких контрольных примерах. Разделяемые этапы можно представить себе как подпрограммы для ваших ручных контрольных примеров. При правильном использовании эта возможность существенно улучшит в долгосрочной перспективе удобство использования ваших контрольных примеров.

Наиболее удобно использовать разделяемые этапы в тех случаях, когда приложение перед выполнением каждого теста приводится в заранее известное состояние. Для тестирования Microsoft Dynamics AX отличной стратегией является запуск приложения из командной строки и его инициализация для тестирования с помощью параметров командной строки. Вот пример того, как запустить Microsoft Dynamics AX и выполнить задание для инициализации данных. Сначала создайте XML-файл *fminitializedata.xml* и сохраните его в хорошо известном каталоге – вы будете указывать имя этого файла в командной строке.

Примечание. Хотя в этом примере используется корневой каталог диска C:, указывать расположение файлов лучше с использованием переменных среды.

```
<?xml version="1.0" ?>
<AxaptaAutoRun
    exitWhenDone="false"
    logFile="c:\AXAutorun.log">
        <Run type="job" name="InitializeFMDataModel" />
</AxaptaAutoRun>
```

Теперь приложение можно запустить с помощью команды Пуск > Выполнить, указав такую командную строку: *ax32.exe -StartUpCmd=AutoRun_c:\fminitializedata.xml*.

Вы могли бы включить эту командную строку вместе с шагами по смене каталога в разделяемый этап «Запустить AX и установить начальный каталог», чтобы контрольный пример всегда начинался из определенного известного каталога, как показано на рис. 15-7.

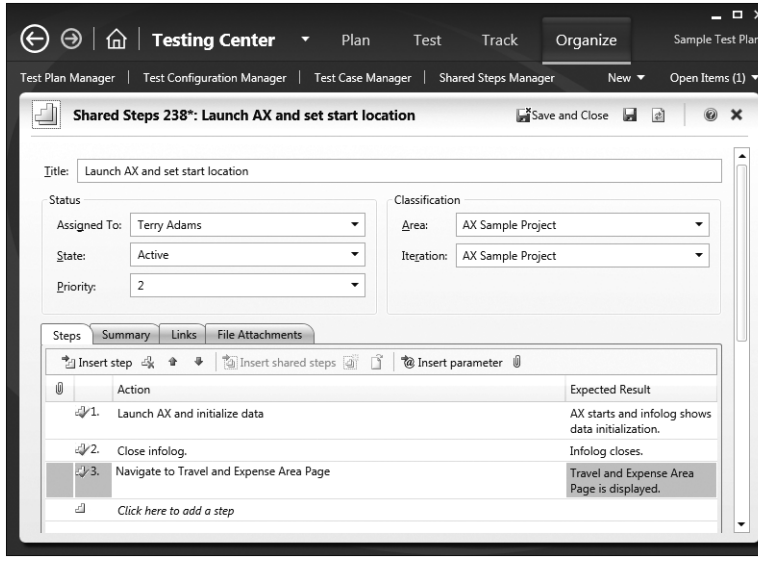

Рис. 15–7. Разделяемый этап

Записывайте разделяемые этапы для последующей «быстрой перемотки»

Microsoft Test Manager включает базовые возможности записи и воспроизведения действий. Ввиду ориентации на ручное тестирование, запись

и воспроизведение не предназначены для полноценной автоматизации тестирования, которое затем можно было бы запускать одним нажатием кнопки. Вместо этого тестировщик может использовать эту функциональность для «быстрой перемотки» рутинной части теста, чтобы перейти к интересной части, где собственно и требуется проверка, а исследовательское тестирование помогает эффективному обнаружению ошибок.

Разделяемые этапы являются отличной отправной точкой в деле повышения эффективности ручного тестирования за счет использования возможностей «быстрой перемотки». Microsoft Test Manager может записывать разделяемые этапы независимо (Organize > Shared Steps Manager > Create Action Recording). Действия, записанные для разделяемого этапа, могут использоваться во всех тестах, где необходимо выполнение этих действий. Вы также можете оптимизировать воспроизведение за счет использования наиболее эффективных и надежных действий.

С учетом длины командной строки и постоянной необходимости быть в известном состоянии, этап «Запустить AX и установить начальный каталог, показанный на рис. 15-7, является хорошим кандидатом на запись. Чтобы выполнять данный этап как можно эффективнее и надежнее, сделайте следующие шаги.

- Чтобы открыть диалоговое окно ввода командной строки, используйте комбинацию клавиш Win+R вместо мыши. В общем случае нажатия комбинаций клавиш являются более подходящими кандидатами для записи и воспроизведения.
- Чтобы перейти на определенную страницу области, введите путь к ней в адресной строке. У этого подхода есть целый ряд преимуществ, потому что он позволяет перейти напрямую к странице области и не зависит от текущего положения в меню, в котором оказалось приложение.

В левой части рис. 15-8 показано, как выглядит Microsoft Test Manager после записи разделяемого этапа, а в правой части показан тест, который использует этот разделяемый этап. Обратите внимание на зеленую стрелку в подсвеченном первом этапе. Она дает вам возможность «быстро перемотать» разделяемый этап.

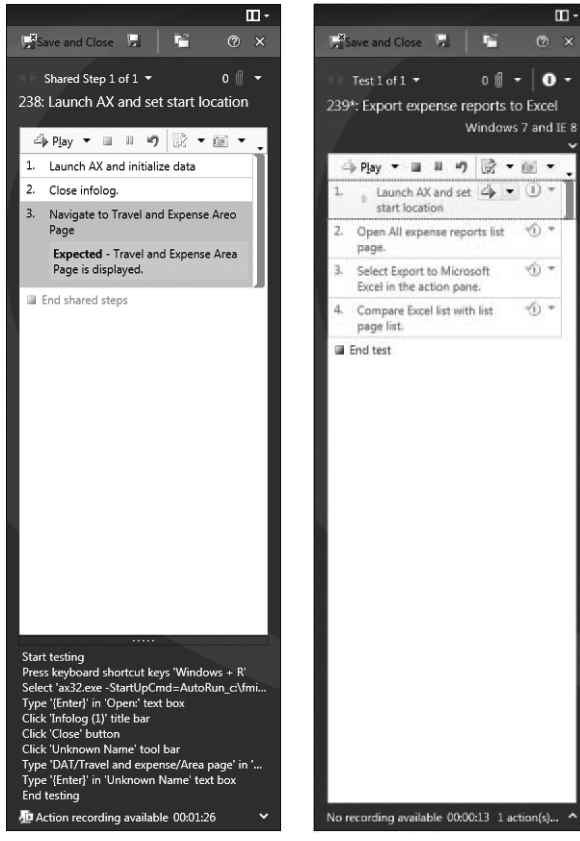

Рис. 15–8. Microsoft Test Manager показывает записанный этап и тест, который его использует

Разрабатывайте тесты эволюционно

Создание детализированных пошаговых контрольных примеров на ранней стадии разработки может стать контрпродуктивным по мере того, как приложение эволюционирует к своей конечной форме. Более удачной альтернативой является разработка контрольных примеров в несколько фаз.

- **Фаза 1.** Определите заголовки тестов, необходимых для планирования требований на вашей текущей фазе разработки. Создайте тесты и связанные метаданные (область, приоритет и т.д.).
- **Фаза 2.** Добавьте конкретику в тесты с использованием подхода на основе намерений. Возможно, для теста нужно создать счет клиента с просроченной задолженностью. Выполнение соответствующих действий может потребовать большого количества отдельных этапов (шагов), но в данной фазе достаточно этапа «Создание счета клиента с просроченной задолженностью».
- **Фаза 3.** По мере необходимости детализируйте тесты. Если ваши тестировщики являются экспертами в предметной области, вам может не потребоваться дополнительная детализация. И хотя при отсутствии деталей появляется изменчивость, запись действия обеспечивает разработчику необходимые детали выполненных шагов. Эта фаза также дает возможность создавать дополнительные этапы, которые могут быть повторно использованы в других тестах. Если для нескольких тестов необходимо «Создать счет клиента с просроченной задолженностью», то создайте разделяемый этап и, возможно, запишите выполняемые в нем действия. В качестве альтернативы вы можете включить в свои данные запись для подходящего клиента.

Используйте ordered test suites для продолжительных сценариев

Тесты с использованием сценариев важны для бизнес-приложений, потому что для бизнес-процессов характерны длинные цепочки передачи работ от одного исполнителя к другому. Отражение этих продолжительных сценариев в виде теста может быть непростой задачей, потому что вряд ли вам захочется иметь тест с десятками этапов (шагов).

Microsoft Test Manager решает эту проблему, предоставляя возможность определять очередность выполнения тестов в тестовом наборе. На рис. 15-9 показан пример законченного сценария для модуля управления персоналом, который разбит на множество коротких тестов, очередность выполнения которых задана в тестовом наборе.

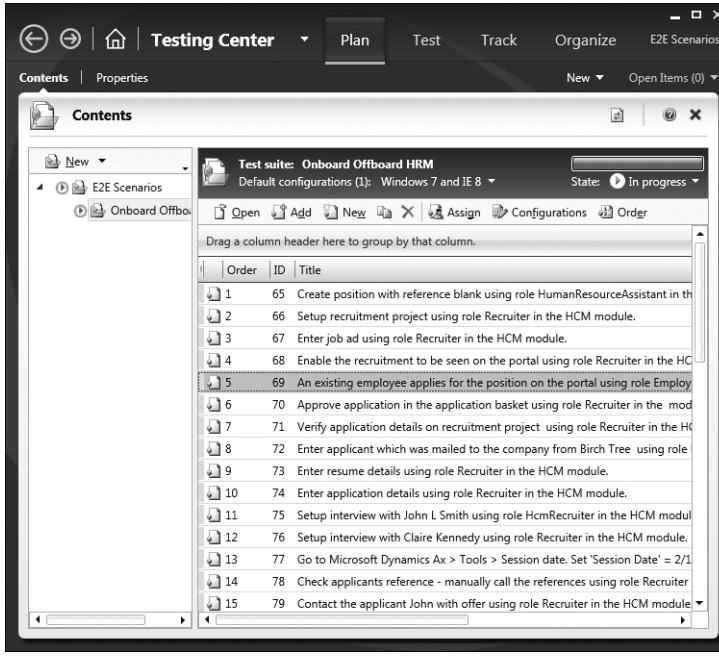

Рис. 15–9. Тестовый набор со множеством тестов

Собираем все элементы вместе

Пока что в этой главе обсуждались ключевые аспекты тестирования разработчиком и функционального тестирования. В данном разделе эти темы объединяются вместе с некоторыми областями управления жизненным циклом.

Выполняйте тесты как часть процесса сборки приложения

Решение Visual Studio 2010 ALM также включает Team Foundation Build – систему поддержки технологических процессов, которую вы можете использовать для компиляции кода, запуска связанных тестов, выполнения анализа кода, осуществления непрерывной сборки и публикации отчетов о выполненных сборках. Вы можете использовать Team Foundation Build на проектах Microsoft Dynamics AX. Хотя процесс сборки выходит за рамки обсуждения этой главы, к ней относится выполнение тестов из Team Foundation Build.

Тесты, выполняемые как часть процесса сборки, должны быть полностью автоматизированы. С учетом этого требования, отправной точкой должны стать тесты, написанные с использованием инфраструктуры SysTest. К счастью, есть инструменты, позволяющие запускать выполнение тестов SysTest Microsoft Dynamics AX из среды Visual Studio. В первую очередь для этого необходимо, чтобы инфраструктура SysTest предоставляла результаты в формате, который ожидает получить Visual Studio, а именно в формате вывода TRX. Тут есть две хорошие новости: во-первых, инфраструктура SysTest предоставляет расширяемую модель для слушателей, обрабатывающих результаты выполнения тестов (т.н. test listeners), во-вторых, компании-партнеры, занимающиеся внедрением Microsoft Dynamics AX, опубликовали на CodePlex пример реализации TRX с использованием возможности Test Listeners. Пакет SysTestListenerTRX для Microsoft Dynamics AX 2012 может быть загружен из раздела *http://dynamicsaxbuild.codeplex.com/releases*.

Вторым шагом является реализация возможности запускать тесты из Visual Studio. В качестве обертки для существующей программы была разработана возможность выполнения произвольных тестов – Generic Test Case. Для данной ситуации это идеально подходит, поскольку Microsoft Dynamics AX может запускать проект тестирования из командной строки и указывать Test Listener и расположение выходного файла.

Допустим, вы хотите выполнить все тесты, помеченные атрибутом *SysTestIntegrationTestAttribute*, который был создан ранее в этой главе. После загрузки и установки пакета SysTestListenerTRX выполните следующие действия.

1. Создайте новый проект тестирования в Microsoft Dynamics AX. Добавьте все тестовые классы, имеющие атрибут *SysTestIntegrationTestAttribute* на самом классе или одном из методов. Как описано ранее в этой главе для check-in тестов, щелкните правой кнопкой мыши по названию проекта и выберите Настройки, а затем выберите в качестве фильтра Интеграционные тесты.
2. Создайте новый проект тестирования в Visual Studio.
3. Выберите пункт меню Test > New Test и затем в диалоговом окне Add New Test дважды щелкните Generic Test.
4. Настройте обобщенный тест, как показано на рис. 15-10, выполнив следующие шаги.

- В поле Specify An Existing Program… укажите полный путь к *Ax32.exe*.
- В поле Command Line Arguments… укажите строку из рис. 15-10. В этой строке указывается, что должен выполняться проект тестирования Dynamic AX с названием «IntegrationTests», что должен использоваться TRX listener и вывод должен быть направлен в файл *%TestOutputDirectory%\AxTestResults.trx*.
- В группе Results Settings поставьте флажок Summary Results File и затем укажите расположение результатов, используя тот же путь и имя файла, что и в командной строке.

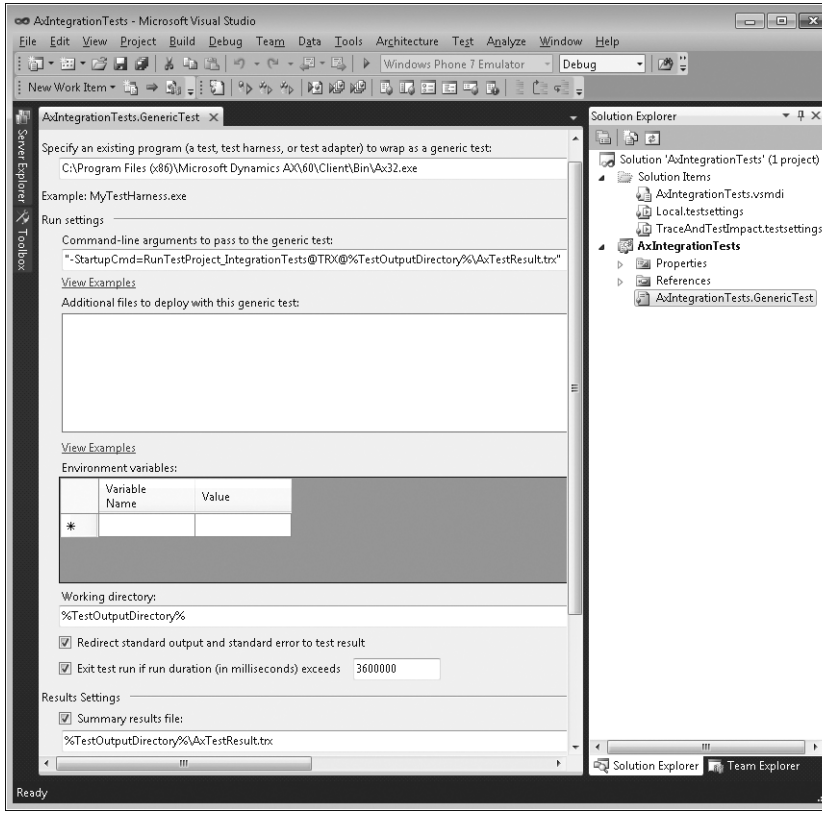

Рис. 15–10. Настройки теста

Когда вы запустите все тесты в решении из Visual Studio (пункт меню Test > Run > All Tests In Solution), то увидите, как откроется и закроется

окно клиента Microsoft Dynamics AX. Результаты для данного примера показаны на рис. 15-11. Два теста были помечены атрибутом *SysTestIntegrationTestAttribute,* и оба отработали успешно.

Используйте правильные тесты для проектов

Обычный проект разработки в Microsoft Dynamics AX использует четыре отдельные среды: разработческую, тестовую, опытную и рабочую. В этом разделе дается краткое описание каждой среды и обсуждается, как применять инструменты тестирования для каждой из них.

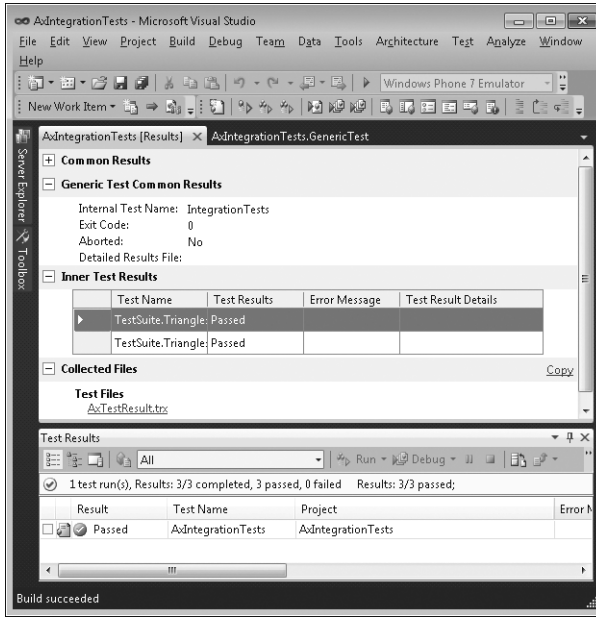

Рис. 15–11. Результаты теста

Разработческая среда – это та, где программисты активно пишут код. Качественный процесс разработки фокусируется на обеспечении качества, настолько близкого к уровню стандартного приложения, насколько это возможно.

Это прекрасная возможность использовать инфраструктуру SysTest, чтобы разрабатывать блочные тесты для новых классов или методов и интеграционные тесты для простых сценариев взаимодействия нескольких классов. Со временем эти автоматизированные тесты смогут сформировать пакет регрессионного тестирования, который будет выполняться во время

процессов возврата модификаций в систему контроля версий и во время сборки, как было описано в этой главе. Процесс ATDD, описанный ранее применительно к проверке требований, тоже должен использоваться в разработческой среде, поэтому тестировщики, задействованные на проекте, также должны быть вовлечены на фазе разработки, предпочтительно используя при этом инструменты тестирования Visual Studio 2010.

Более обширное тестирование проводится в тестовой среде. Для этой среды характерны различные уровни интеграционного тестирования, при этом уделяется особое внимание тому, чтобы бизнес-процессы функционировали от начала до конца. Хорошим подходом здесь будет создание тестовых наборов, использующих выполняющиеся в определенной последовательности тесты из Microsoft Test Manager. Это является прекрасной возможностью постепенно детализировать тесты, используя хорошо продуманный подход с разделяемыми этапами для минимизации дублирования в различных тестовых наборах. По мере изменения продукта и создания новых сборок должен выполняться набор регрессионных тестов SysTest.

Основной деятельностью в опытной среде является приемочное тестирование (user acceptance testing, UAT). Наборы тестов Microsoft Test Manager, разработанные для тестовой среды, могут сформировать основу для приемочного тестирования, выполняемого бизнес-пользователями. В этой среде в качестве данных должен использоваться слепок с рабочей базы.

Если в предыдущих средах все прошло хорошо, то код переносится в рабочую среду. Для минимизации времени простоев здесь после развертывания нового кода проводится лишь поверхностное тестирование («дымовые тесты»). Обычно это ручные тесты, определенные бизнес-пользователями, но выполняемые ИТ-специалистами, занимающимися развертыванием. Опять-таки, вы можете использовать Microsoft Test Manager для определения тестов и обеспечить для аудита и отладки среду выполнения с записанными этапами тестирования.

Чтобы убедиться, что у вас есть качественный и всеобъемлющий план тестирования, вам, вероятно, захочется ознакомиться с другой документацией, содержащей описания процессов и рекомендации по разработке, сфокусированной на качестве. Более подробную информацию вы можете найти в документе «Testing Best Practices» для версии Microsoft Dynamics AX 2012, доступном по адресу: *http://www.microsoft.com/download/en/details.aspx?id=27565,* а также в описании методологии Microsoft Dynamics SureStep по адресу: *http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=5320*.