Лабораторная работа

Юнит-тестирование, покрытие кода и анализ похожего кода с помощью Visual Studio 2013

Lab version: 12.0.30723.00 Update 3

Last updated: 12/12/2013

СОДЕРЖАНИЕ

Резюме

Из этой лабораторной работы вы изучите расширяемый адаптерами типа Nunit и xUnit.net движок юнит-тестирования в Visual Studio 2013, а также покрытие кода и возможности анализа похожего кода.

Prerequisites

Для выполнения лабораторной работы вам понадобится виртуальная машина с Visual Studio 2013. Подробнее про то, где загрузить и как ее использовать, здесь.

О компании Fabrikam Fiber

Эти лабораторные работы в качестве основы для сценариев, о которых вы узнаете в процессе, оперируют несуществующей компанией Fabrikam Fiber. Fabrikam Fiber занимается кабельным телевидением и сопутствующими сервисами в США. Компания быстро растет и уже начала использовать Microsoft Azure для того, чтобы масштабировать свой веб-сайт для обслуживания их запросов и отслеживания деятельности инженеров. Компания использует локальное приложение ASP.NET MVC для управления заказами клиентов.

В этих лабораторных работах вы изучите сценарии, включенные в рабочий процесс команды разработки и тестирования Fabrikam Fiber. Команда, состоящая из 8-10 человек, решила использовать средства управления жизненным циклом проектов Visual Studio 2013 для того, чтобы контролировать программный код, выполнять сборки, тестировать веб-сайты, планировать и отслеживать происходящее с проектом.

Упражнения

Эта лабораторная работа включает в себя следующие упражнения:

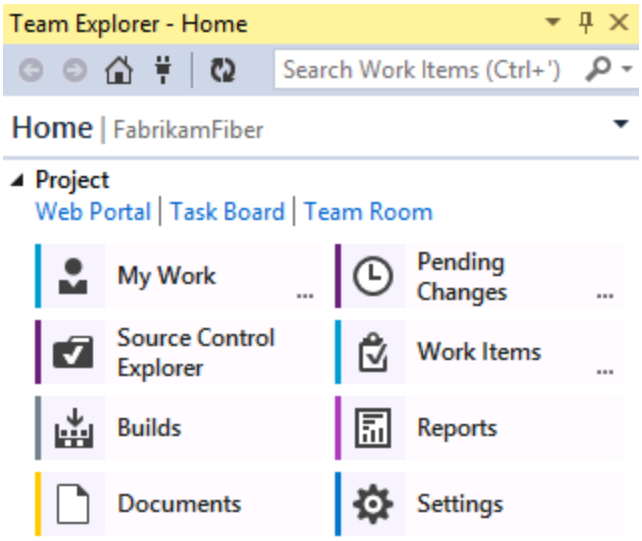
1. Юнит-тестирование
2. Покрытие кода
3. Анализ похожего кода

Примерное время выполнения лабораторной работы: **30 минут**.

Упражнение 1: юнит-тестирование

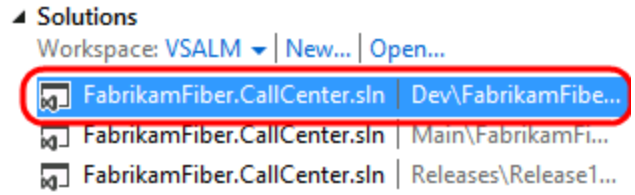
В этом упражнении вы узнаете о нововведениях юнит-тестирования в Visual Studio 2013.

1. Войдите под аккаунтом **Julia Ilyiana** (VSALM\Julia). Пароль: **P2ssw0rd.**
2. Запустите **Visual Studio 2013** и откройте **Team Explorer**. Вы должны быть подключены к командному проекту FabrikamFiber, если этого не произошло, нажмите **Connect to Team Projects** ( ) и инициируйте подключение.



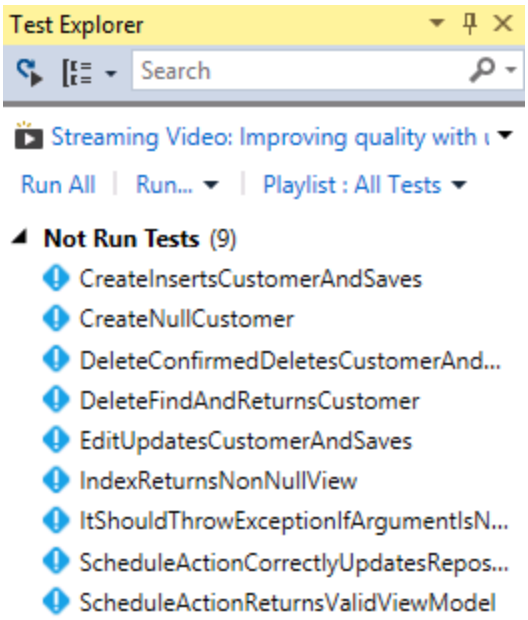
Изображение 1
Team Explorer - Home

3. В **Team Explorer – Home нажмите два раза** на первом решении **FabrikamFiber.CallCenter.sln**.



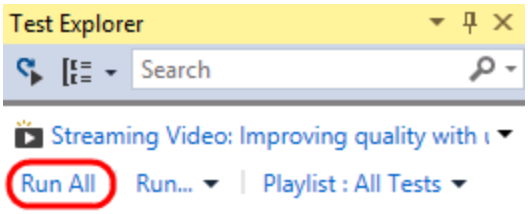
Изображение 2
Проект Fabrikam Fiber

4. Откройте **Test Explorer** из **Test | Windows | Test Explorer**. Состояние обнаруженных тестов по умолчанию выставлено в Not Run.



Изображение 3
Test Explorer

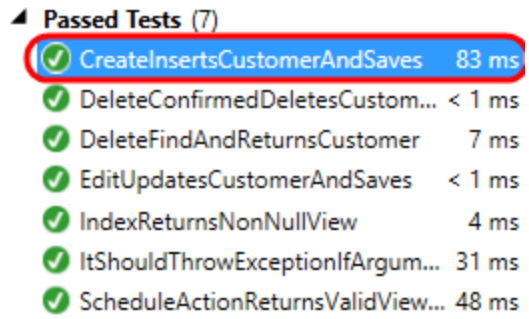
5. Нажмите на **Run All** для запуска всех юнит-тестов.



Изображение 4
Запуск всех юнит-тестов

> **Примечание:** Visual Studio содержит функцию **Continuous Test Runner,** которая может быть включена с помощью опции **Test | Test Settings | Run Tests After Build**. Эта функция позволит тестам всегда выполняться после сборок.

6. Разверните **Passed Tests** и **нажмите два раза** на тесте **CreateInsertsCustomerAndSaves** для открытия исходного кода.



Изображение 5

Метод теста, использующего MSTest

7. В файле *CustomersControllerTest.cs*, обратите внимание на тестовый метод, помеченный как юнит-тест атрибутом TestMethod, использующимся MSTest.

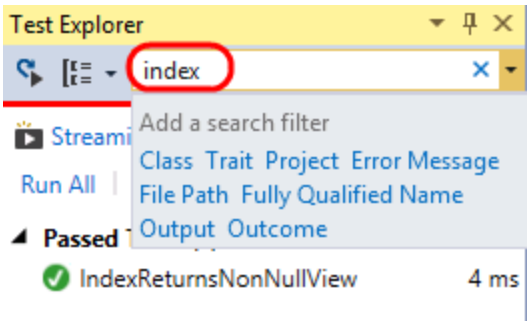
```
[TestMethod()]
0 references | Julia Ilyiana | 1 change
public void CreateInsertsCustomerAndSaves()
{
    controller.Create(new Customer());

    Assert.IsTrue(mockCustomerRepo.IsInsertOrUpdateCalled);
    Assert.IsTrue(mockCustomerRepo.IsSaveCalled);
}
```

Изображение 6

Код юнит-теста MSTest

> **Примечание:** индикатор состояния теста над определением метода называется индикатором CodeLens. Этот индикатор сообщает о том, что последний запуск теста был успешным. Подробнее про CodeLens – в лабораторной работе "*New Collaboration Experiences for Development Teams using Team Foundation Server 2013*".

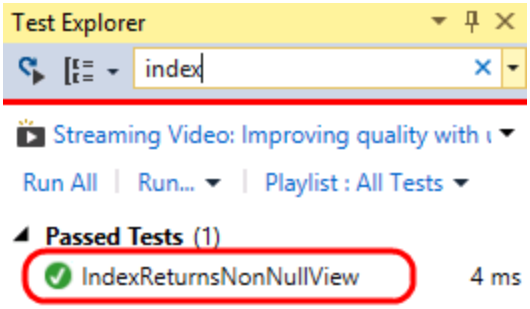
8. Введите index в текстовое поле **Search** в верхней части Test Explorer.



Изображение 7

Поиск в Test Explorer

9. В найденных поиском результатах **нажмите два раза** на тесте **IndexReturnsNonNullView**.



Изображение 8

Метод теста, использующего Xunit

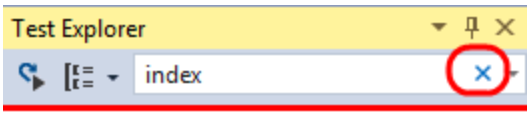
10. В *HomeControllerTest.cs* можно видеть, что тест **IndexReturnsNonNullView** использует фреймворк тестирования **XUnit**. Последний адаптер XUnit для Visual Studio 2013 можно найти в галерее Visual Studio http://aka.ms/UnitTestAdapters.

```
[TestFixture]
0 references | Julia Ilyiana | 1 change
public class HomeControllerTest
{
    [Xunit.Fact]
    | 0 references | Julia Ilyiana | 1 change
    public void IndexReturnsNonNullView()
```

Изображение 9

Пример теста Xunit в Visual Studio 2013

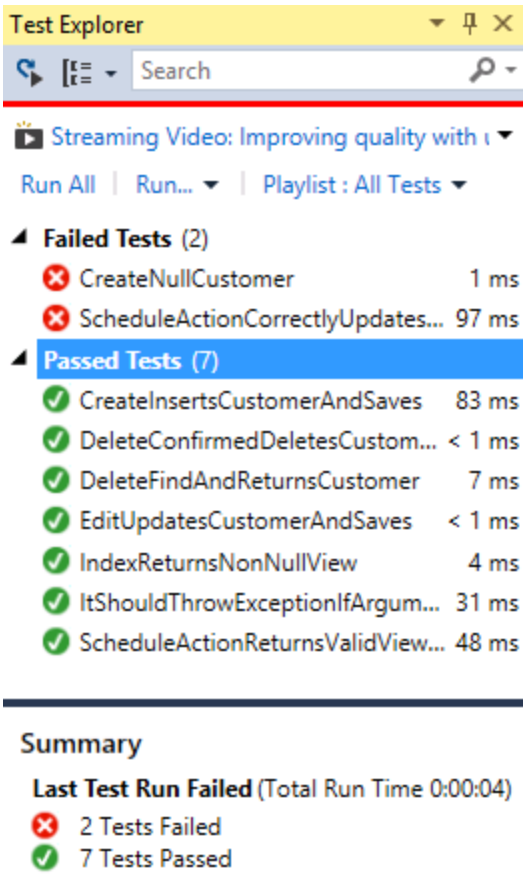
11. Нажмите на **X** в текстовом поле поиска в Test Explorer.



Изображение 10

Кнопка X

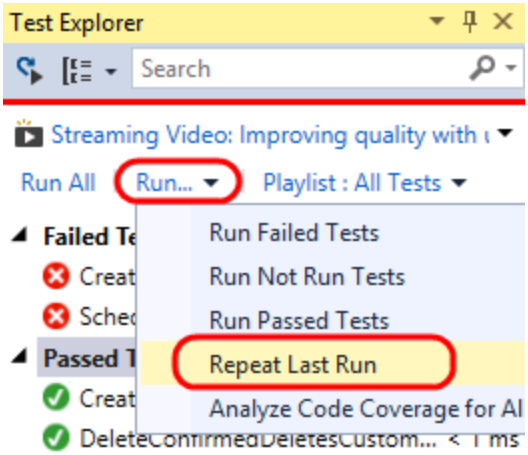
12. Найденные результаты по умолчанию группируются по результатам тестов, и имеют время выполнения.



Изображение 11

Результаты выполнения теста

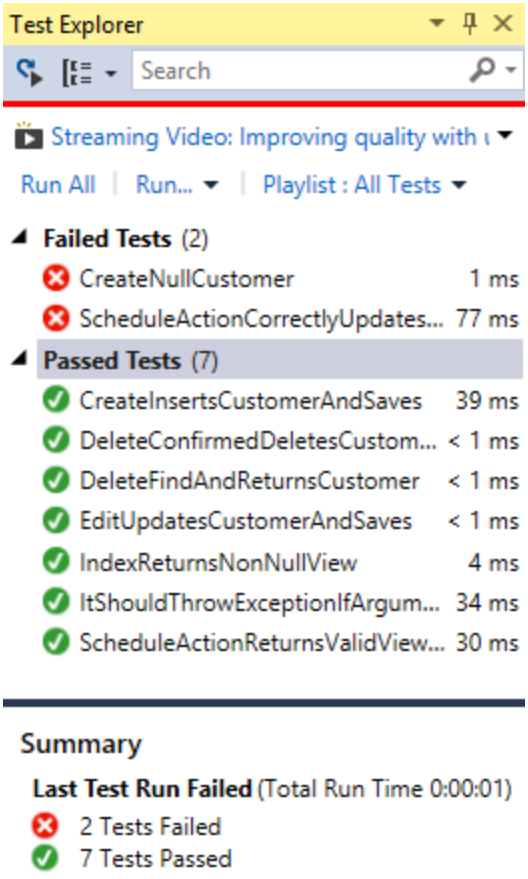
13. В последних версиях внимание было уделено и производительности выполнения тестов. Изучите время выполнения и нажмите на **Run...** и **Repeat Last Run**. Обратите внимание на разницу во времени выполнения.



Изображение 12

Повтор последнего запуска теста

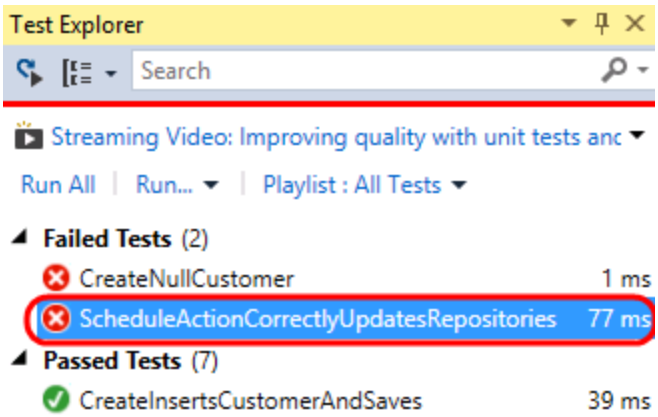
> **Примечание:** у вас будет иное время выполнения. Первый запуск выполняется медленнее, чем дальнейшие, так как тесты и движок должны загрузиться.



Изображение 13

Второй запуск теста без изменений выполняется быстрее

14. Нажмите на невыполнившемся тесте “**ScheduleActionCorrectlyUpdatesRepositories**” для просмотра его результата.

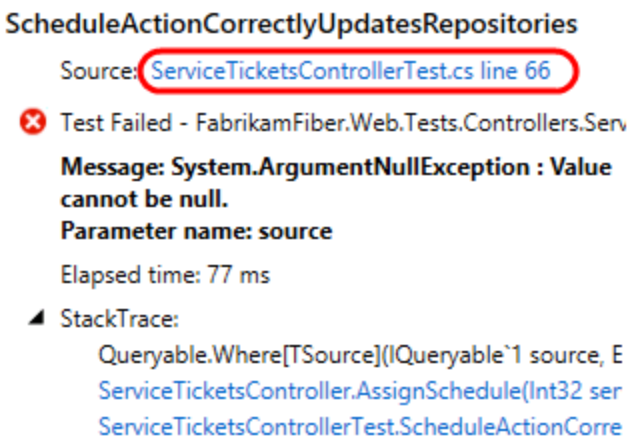


Изображение 14

Невыполнившийся результат

> **Примечание:** вы можете нажать правой кнопкой на результатах тестов и нажать Copy для копирования информации в буфер обмена, что может быть полезно при отправке по E-mail, например.

15. Обзор теста показывает, что во время теста было выброшено исключение **ArgumentNullException**, и содержит стектрейс. Мы можем перейти прямо в код теста или на место внутри стектрейса. **Нажмите на** ссылке на исходный код.



Изображение 15

Переход в исходный код теста

```
[Xunit.Fact]
0 references | Julia Ilyiana | 1 change
public void ScheduleActionCorrectlyUpdatesRepositories()
{
    this.SetupController();

    // Arrange
    var scheduleItems = new List<ScheduleItem>();
    scheduleItems.Add(new ScheduleItem { ServiceTicketID = 1 });
    //mockScheduleItemRepo.SetReturnValue("get_All", scheduleItems.AsQuer
    mockScheduleItemRepo.SetReturnValue("InsertOrUpdate", null);
```

Изображение 16

Исходный код метода

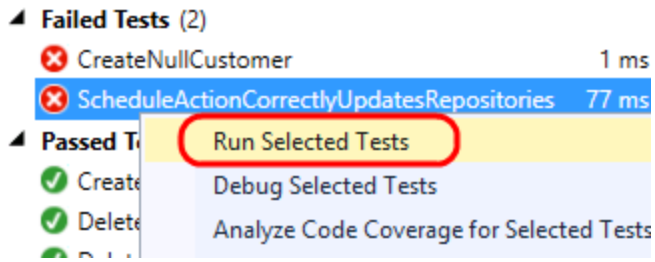
16. Раскомментируйте закомментированную строку кода. Предположим, что она приводит к ошибке.

```
[Xunit.Fact]
0 references | Julia Ilyiana | 1 change
public void ScheduleActionCorrectlyUpdatesRepositories()
{
    this.SetupController();

    // Arrange
    var scheduleItems = new List<ScheduleItem>();
    scheduleItems.Add(new ScheduleItem { ServiceTicketID = 1 });
    mockScheduleItemRepo.SetReturnValue("get_All", scheduleItems.AsQuerya
    mockScheduleItemRepo.SetReturnValue("InsertOrUpdate", null);
```

Изображение 17

Исправление ошибки

17. Нажмите **Ctrl+S**.
18. **Нажмите правой кнопкой** на непрошедшем тесте в Test Explorer и нажмите **Run Selected Tests**.

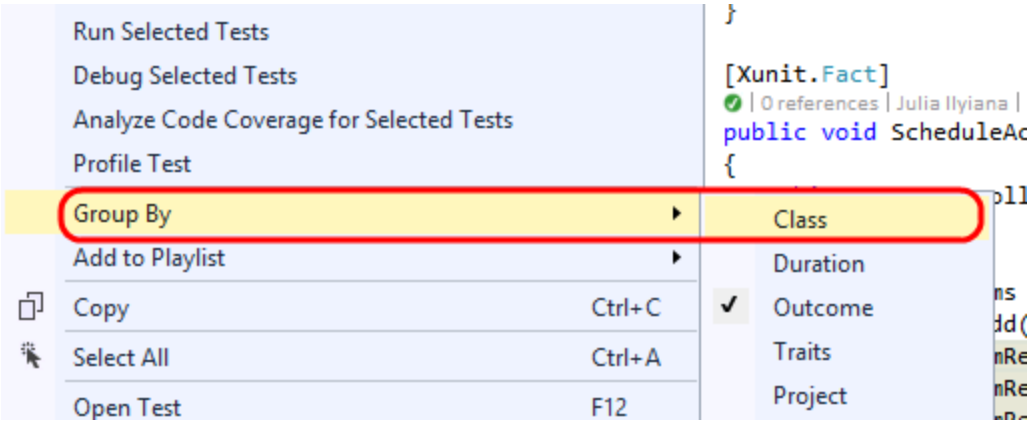


Изображение 18

Запуск теста для проверки исправления

19. Мы увидели, как запускать тесты, искать их и группировать по результатам выполнения. Посмотрим на то, как организовать и ориентироваться в юнит-тестах.

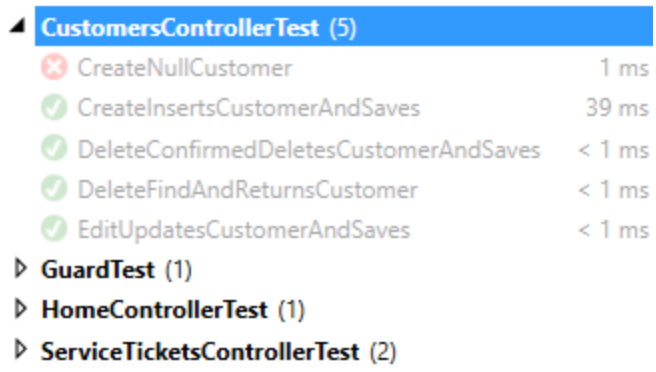
20. Начиная с Visual Studio 2012 Update 2 есть несколько полезных опций. **Нажмите правой кнопкой** в окне Test Explorer и нажмите на **Group By | Class**.



Изображение 19

Группировка по классу

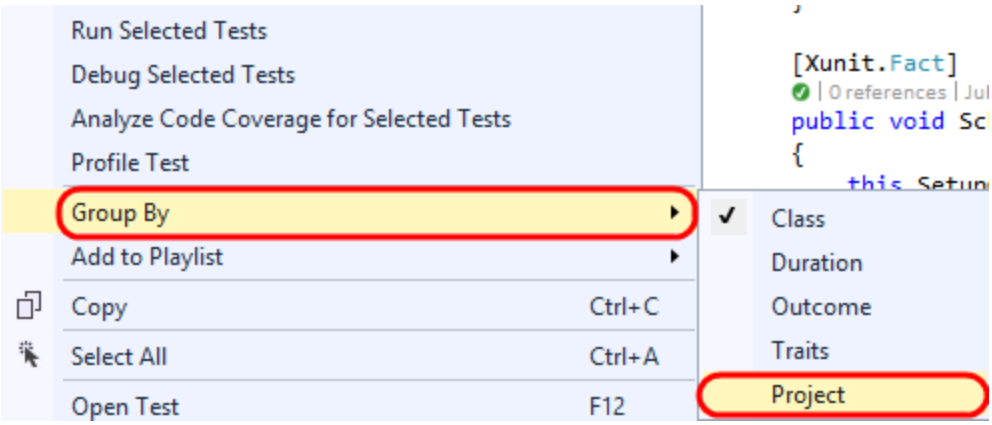
21. Предположим, что тесты хорошо организованы и имеют понятные имена классов. Вы можете в этом случае просто найти и запустить тесты для класса CustomersControllerTest.



Изображение 20

Группировка тестов по классам

22. **Нажмите правой кнопкой** в окне Test Explorer и нажмите на **Group By | Project**.



Изображение 21

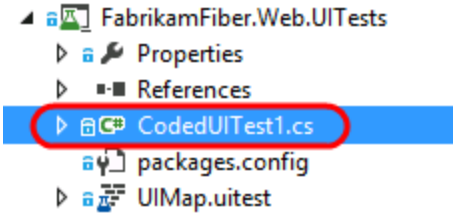
Группировка тестов по проекту

23. Группировка тестов по проекту может быть полезна для навигации по тестам на уровне проектов.

Изображение 22

Группировка тестов по проекту

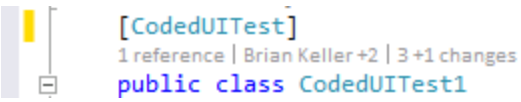
24. Начиная с Visual Studio 2012 Update 1, вы можете использовать так называемые метки (traits) для настраиваемой группировки. Предположим, что надо объединить все тесты coded UI в одну группу. Откройте **CodedUITest1.cs** из проекта **FabrikamFiber.Web.UITests**.



Изображение 23

Код теста

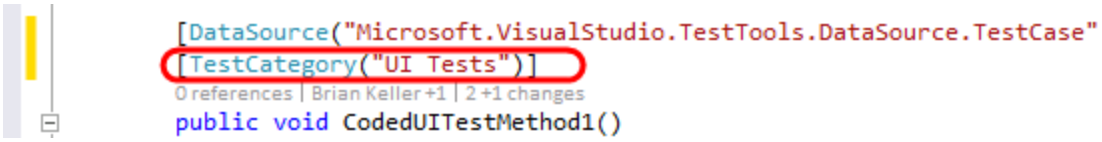
25. Раскомментируйте атрибут **CodedUITest** в определении класса CodedUITest1.



Изображение 24

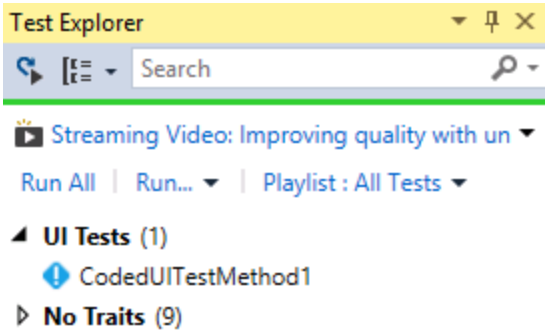
Атрибут CodedUITest

26. Добавьте атрибут **TestCategory** в определение метода CodedUITestMethod1 со значением "**UI**".



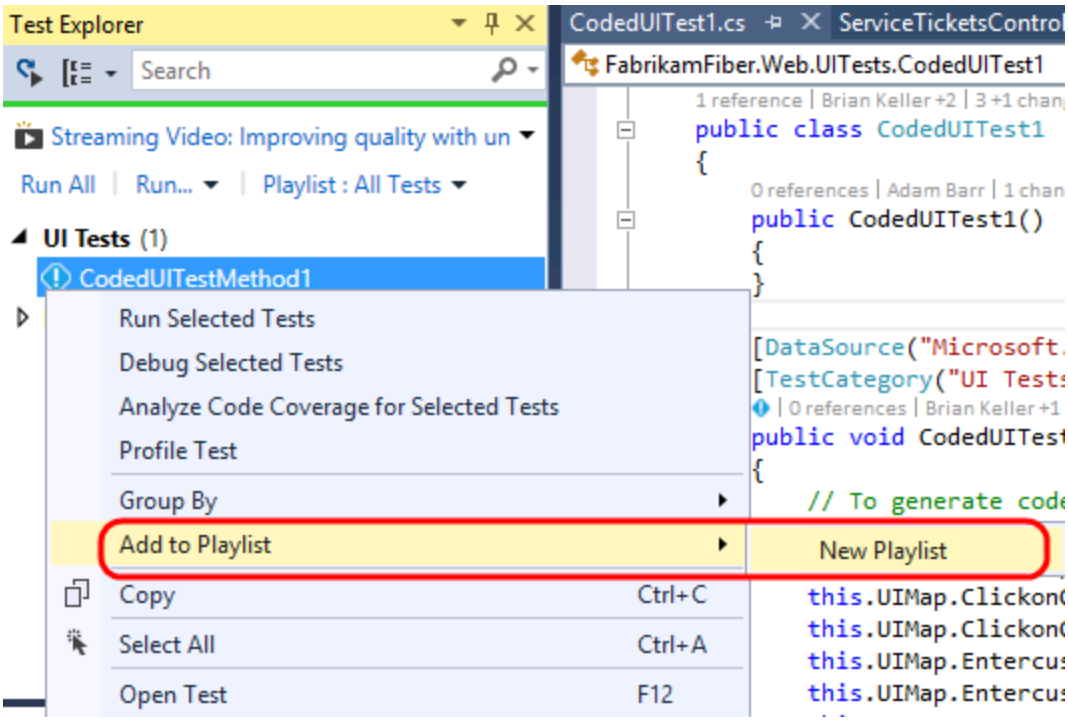
Изображение 25
Метка метода теста

27. Пересоберите решение, нажав **Ctrl+Shift+B**.
28. **Нажмите правой кнопкой** в Test Explorer и выберите **Group By | Traits**.
29. Теперь, когда все тесты coded UI объединены в категорию, найти и запустить нужные тесты гораздо проще.



Изображение 26
Окно Test Explorer со сгруппированными тестами

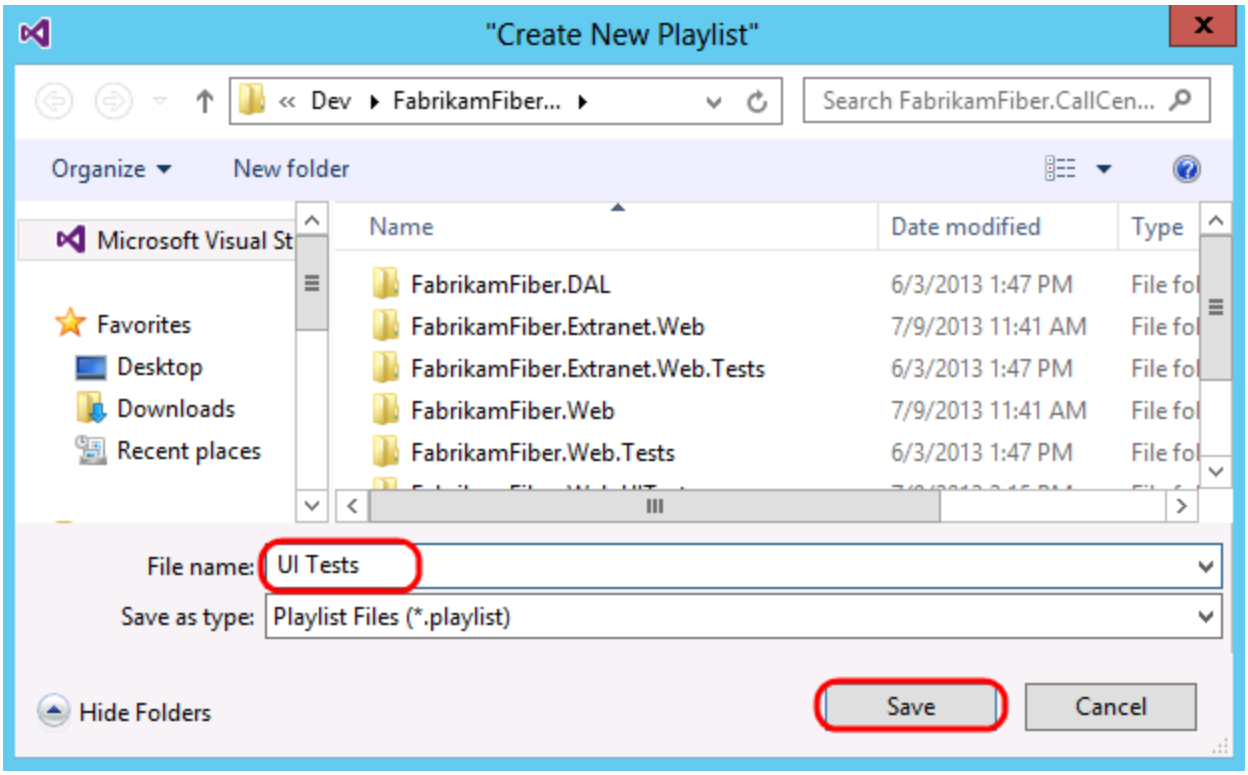
30. Начиная с Visual Studio 2012 Update 2 вы можете создавать наборы тестов, или плейлисты, что дает возможность настраивать группировку без модификации кода юнит-тестов. **Нажмите правой кнопкой** на тесте ***CodedUITestMethod1*** и выберите **Add to Playlist | New Playlist**.



Изображение 27

Создание плейлиста тестов

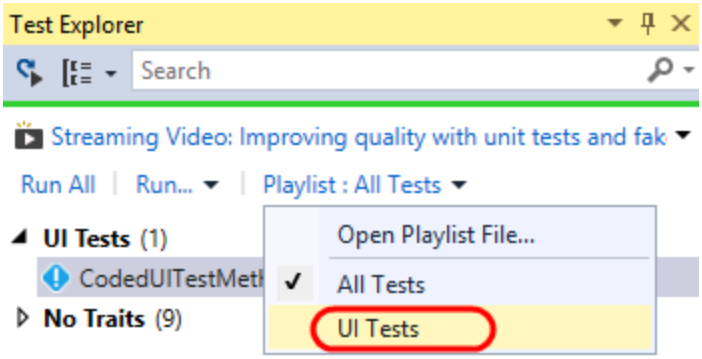
31. Введите название файла “**UI Tests**” и нажмите на **Save**.



Изображение 28

Сохранение плейлиста

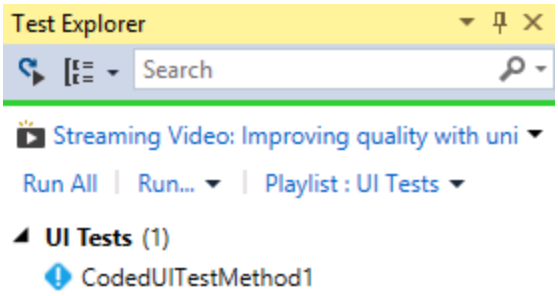
32. Нажмите на выпадающем списке **Playlist** и нажмите на плейлисте “**UI Tests**”.



Изображение 29

Выбор плейлиста

33. Теперь в Test Explorer видны только те тесты, которые относятся к плейлисту, и это позволяет быстрее и проще производить навигацию по ним.



Изображение 30

Выбранный плейлист

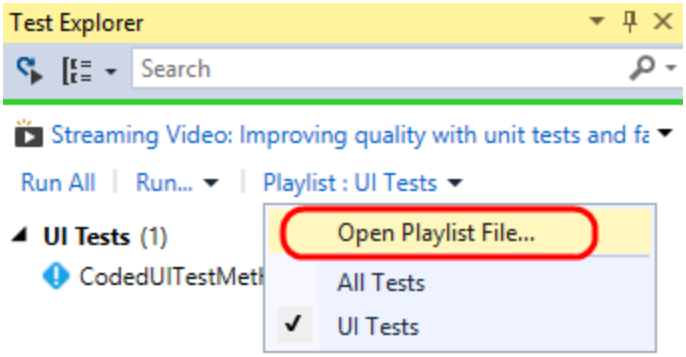
34. Плейлисты основаны на простых XML-файлах, определяющих тесты, которые должны быть включены в плейлисты. Ниже приведено содержимое файла для плейлиста UI Tests.

```
<Playlist Version="1.0">
  <Add Test="FabrikamFiber.Web.UITests.CodedUITest1.CodedUITestMethod1" />
</Playlist>
```

Изображение 31

Содержимое плейлиста теста

35. Плейлистами можно делиться с другими членами команды через командный сайт, по E-mail или через систему контроля версий. Для того, чтобы открыть файл плейлиста, нажмите на **Playlist** и выберите **Open Playlist File**.

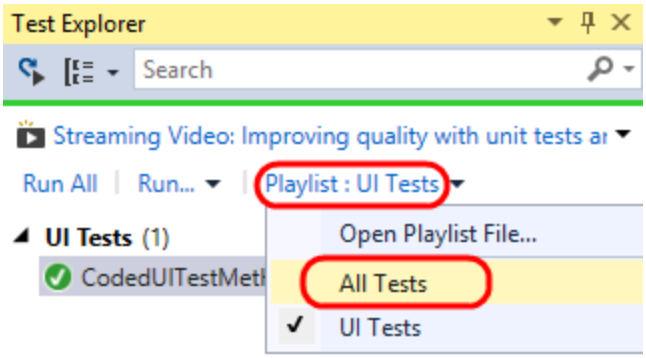


Изображение 32
Открытие плейлиста

Упражнение 2: покрытие кода

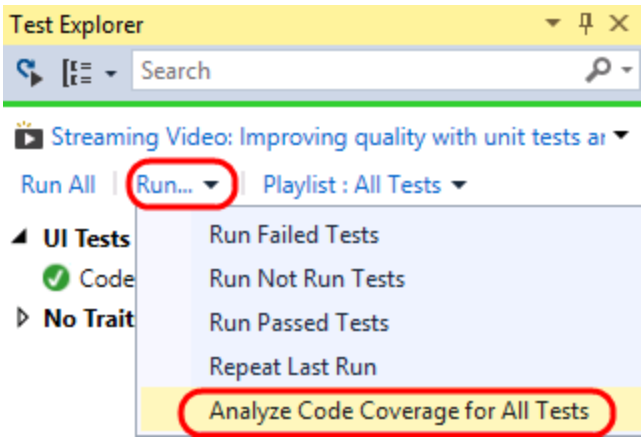
В этом упражнении вы узнаете об изменениях покрытия кода и других нововведениях Visual Studio 2013, позволяющих проще интегрироваться в цикл разработки. Функция покрытия кода доступна в Visual Studio Premium и Ultimate.

1. Нажмите на выпадающее окно **Playlist** и выберите **All Tests**.



Изображение 33
Выбор плейлиста All Tests

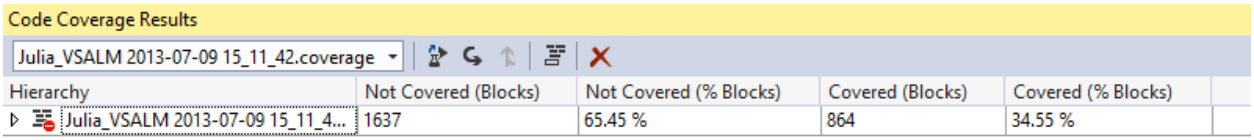
2. Для анализа покрытия кода для всех тестов нажмите на **Run** и на **Analyze Code Coverage for All Tests**. Это запустит процесс сборки, тестирования и сборки результатов покрытия кода.



Изображение 34

Анализ покрытия кода

3. Результаты можно посмотреть в окне **Code Coverage Results**. Обратите внимание на скриншот – по умолчанию покрытие измеряется в блоках кода, где блок кода – это код, содержащий один вход и один выход.



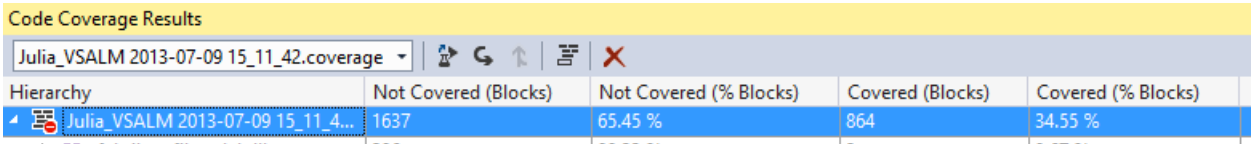
Hierarchy	Not Covered (Blocks)	Not Covered (% Blocks)	Covered (Blocks)	Covered (% Blocks)
Julia_VSALM 2013-07-09 15_11_4...	1637	65.45 %	864	34.55 %

Изображение 35

Результаты анализа покрытия кода

> **Примечание:** если необходимо увидеть результаты в виде количества строк кода, нажмите правой кнопкой на окне Code Coverage Results и выберите Add/Remove Columns.

4. Разверните основную ветку в результатах. По умолчанию мы видим все загруженные во время теста сборки, для которых доступны файлы .pdb.



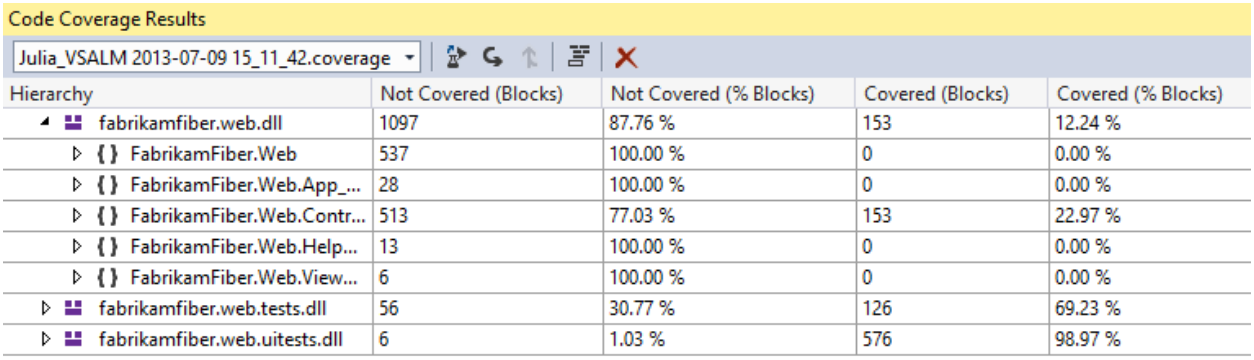
Hierarchy	Not Covered (Blocks)	Not Covered (% Blocks)	Covered (Blocks)	Covered (% Blocks)
Julia_VSALM 2013-07-09 15_11_4...	1637	65.45 %	864	34.55 %
fabrikamfiber.dal.dll	296	99.33 %	2	0.67 %
fabrikamfiber.extranet.web.dll	171	98.28 %	3	1.72 %
fabrikamfiber.extranet.web.te...	11	73.33 %	4	26.67 %
fabrikamfiber.web.dll	1097	87.76 %	153	12.24 %
fabrikamfiber.web.tests.dll	56	30.77 %	126	69.23 %
fabrikamfiber.web.uitests.dll	6	1.03 %	576	98.97 %

Изображение 36

Результаты покрытия кода по сборкам

Примечание: вы можете настраивать выборку сборок, внеся изменения в файл .runsettings. Подробнее - Customizing Code Coverage Analysis.

5. Разверните **fabrikamfiber.web.dll** для просмотра покрытия кода в этой сборке.


Code Coverage Results

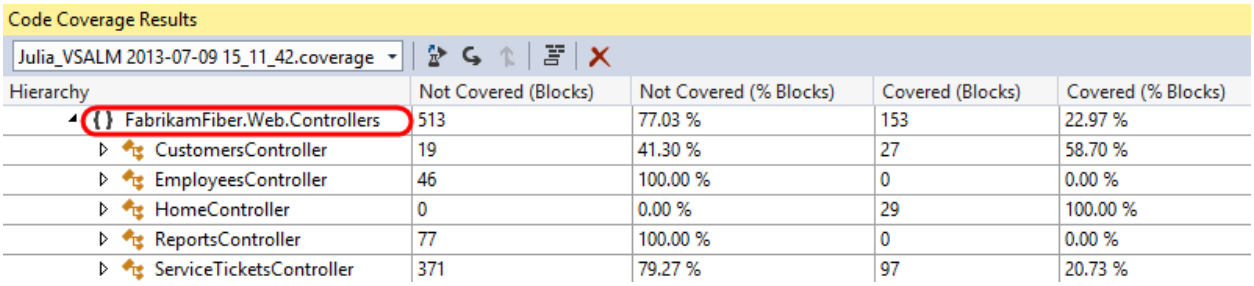
Julia_VSALM 2013-07-09 15_11_42.coverage

Hierarchy	Not Covered (Blocks)	Not Covered (% Blocks)	Covered (Blocks)	Covered (% Blocks)
fabrikamfiber.web.dll	1097	87.76 %	153	12.24 %
FabrikamFiber.Web	537	100.00 %	0	0.00 %
FabrikamFiber.Web.App_...	28	100.00 %	0	0.00 %
FabrikamFiber.Web.Contr...	513	77.03 %	153	22.97 %
FabrikamFiber.Web.Help...	13	100.00 %	0	0.00 %
FabrikamFiber.Web.View...	6	100.00 %	0	0.00 %
fabrikamfiber.web.tests.dll	56	30.77 %	126	69.23 %
fabrikamfiber.web.uitests.dll	6	1.03 %	576	98.97 %

Изображение 37

Результаты анализа покрытия кода внутри пространства имен сборки

6. Разверните пространство имен **FabrikamFiber.Web.Controllers** и изучите покрытие кода по классам. Видно, что класс HomeController покрыт хорошо, класс EmployeesController не покрыт вообще.


Code Coverage Results

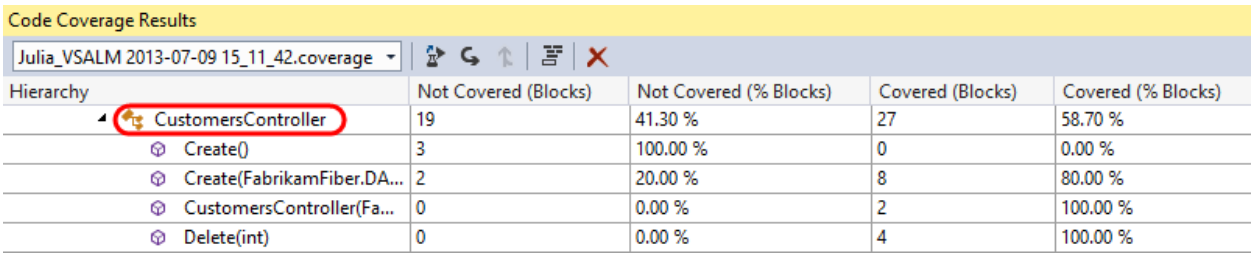
Julia_VSALM 2013-07-09 15_11_42.coverage

Hierarchy	Not Covered (Blocks)	Not Covered (% Blocks)	Covered (Blocks)	Covered (% Blocks)
FabrikamFiber.Web.Controllers	513	77.03 %	153	22.97 %
CustomersController	19	41.30 %	27	58.70 %
EmployeesController	46	100.00 %	0	0.00 %
HomeController	0	0.00 %	29	100.00 %
ReportsController	77	100.00 %	0	0.00 %
ServiceTicketsController	371	79.27 %	97	20.73 %

Изображение 38

Результаты анализа покрытия кода для классов внутри пространства имен

7. Посмотрим на классы, развернув класс **CustomersController**.


Code Coverage Results

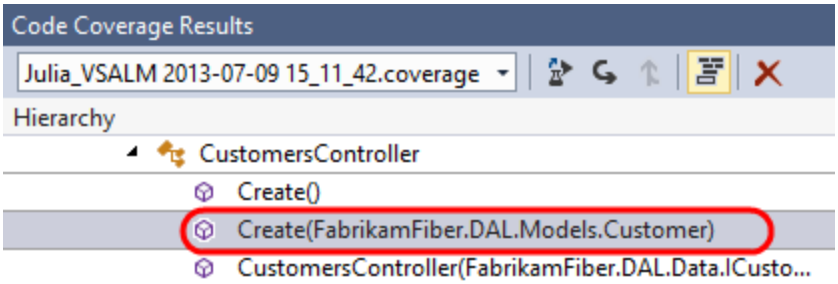
Julia_VSALM 2013-07-09 15_11_42.coverage

Hierarchy	Not Covered (Blocks)	Not Covered (% Blocks)	Covered (Blocks)	Covered (% Blocks)
CustomersController	19	41.30 %	27	58.70 %
Create()	3	100.00 %	0	0.00 %
Create(FabrikamFiber.DA...	2	20.00 %	8	80.00 %
CustomersController(Fa...	0	0.00 %	2	100.00 %
Delete(int)	0	0.00 %	4	100.00 %

Изображение 39

Результаты анализа покрытия кода для методов

8. **Нажмите два раза** на конструкторе **Create(FabrikamFiber.DAL.Models.Customer)** для навигации по исходному коду для визуализации покрытия блока.



Изображение 40

Переход к исходному коду

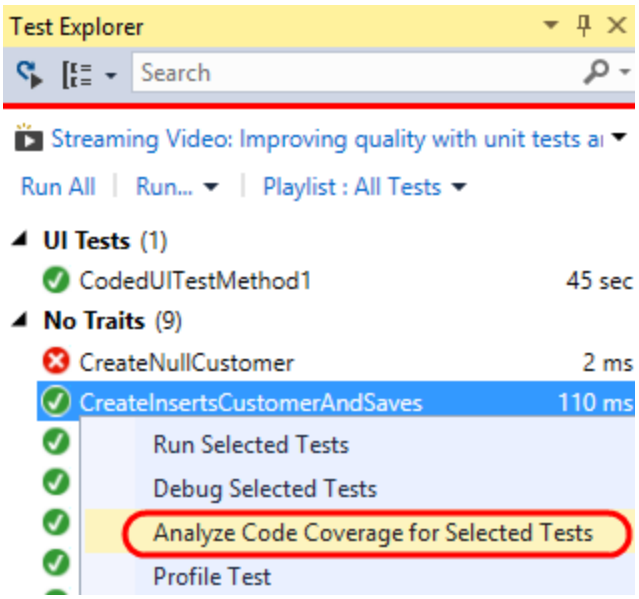
9. Код, подсвеченный синим, это блок, покрытый тестами, красным – не покрытый.

```
[HttpPost]
2 references | 1/2 passing | Brian Keller +2 | 4 changes
public ActionResult Create(Customer customer)
{
    //check model state
    if (ModelState.IsValid)
    {
        this.customerRepository.InsertOrUpdate(customer);
        this.customerRepository.Save();
        return RedirectToAction("Index");
    }

    return this.View();
}
```

Изображение 41

Подсветка кода согласно его покрытию тестами

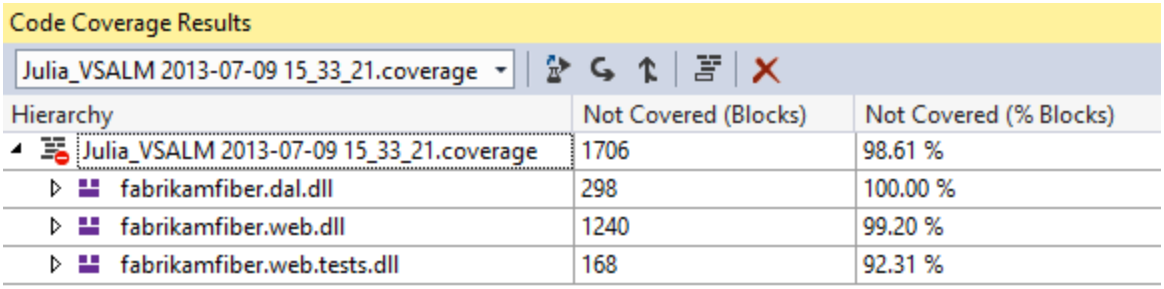
10. Можно также посмотреть покрытие кода для выборки тестов. В Test Explorer нажмите на методе **CreatInsersCustomerAndSaves**, **нажмите на нем правой кнопкой** и выберите **Analyze Code Coverage for Selected Tests**.



Изображение 42

Анализ покрытия кода для выборки тестов

11. После завершения выполнения теста разверните основной узел – загружаемые сборки показываются со статистикой.

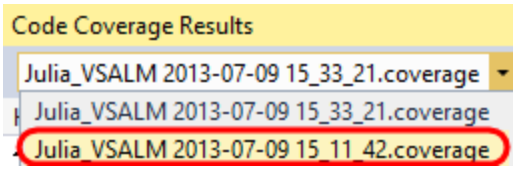


Hierarchy	Not Covered (Blocks)	Not Covered (% Blocks)
Julia_VSALM 2013-07-09 15_33_21.coverage	1706	98.61 %
fabrikamfiber.dal.dll	298	100.00 %
fabrikamfiber.web.dll	1240	99.20 %
fabrikamfiber.web.tests.dll	168	92.31 %

Изображение 43

Результаты анализа покрытия кода

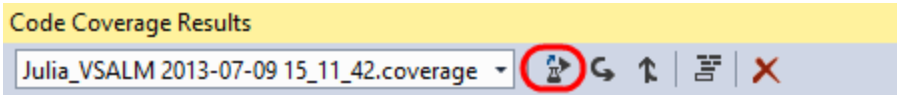
12. Ориентироваться по результатам покрытия кода можно также с помощью выпадающего меню в окне Code Coverage Results. Нажмите на первом файле с анализом покрытия кода.



Изображение 44

Выбор конкретного набора результатов анализа покрытия кода

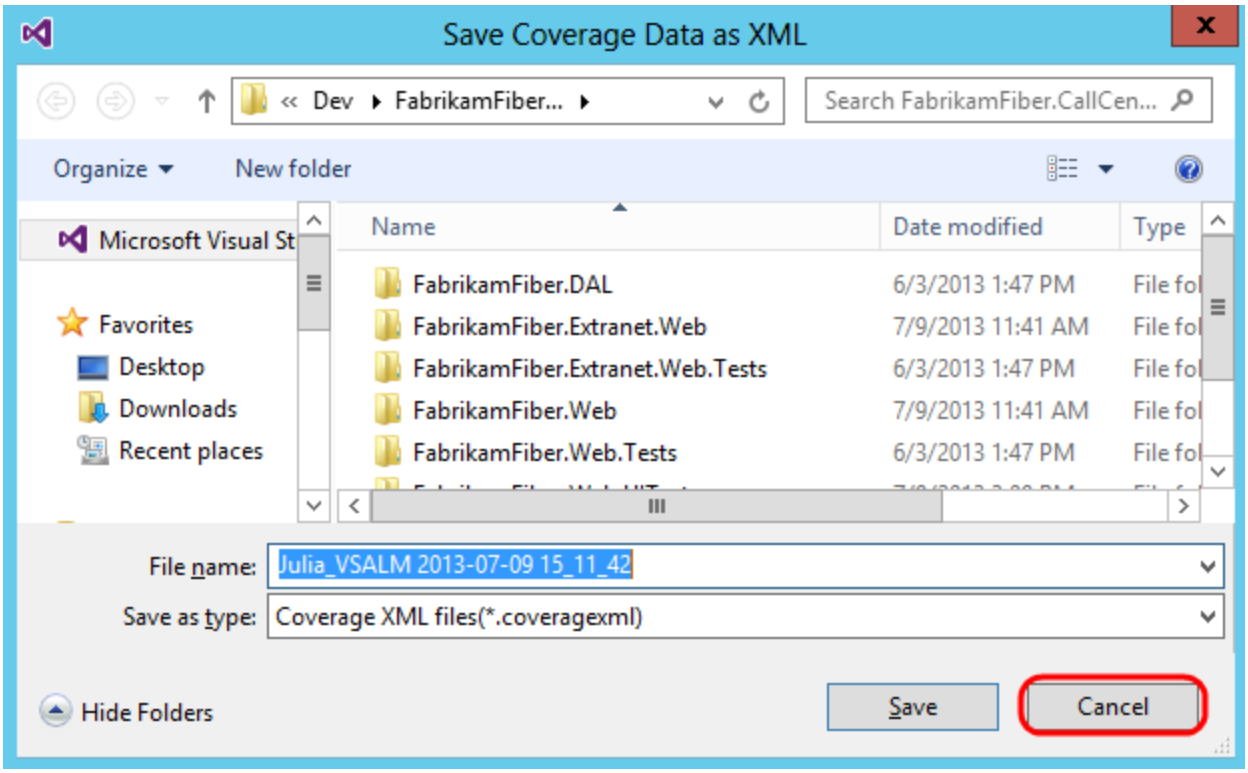
13. Предположим, что нам надо использовать результаты анализа покрытия кода для отчета или просто поделиться ими с кем-нибудь внешним. Нажмите на **Export Results** .



Изображение 45

Экспортирование результатов анализа покрытия кода

14. В **Save Coverage Data as XML** можно сохранить данные по анализу в файл XML. В рамках этой лабораторной работы делать этого не надо, нажмите Cancel.



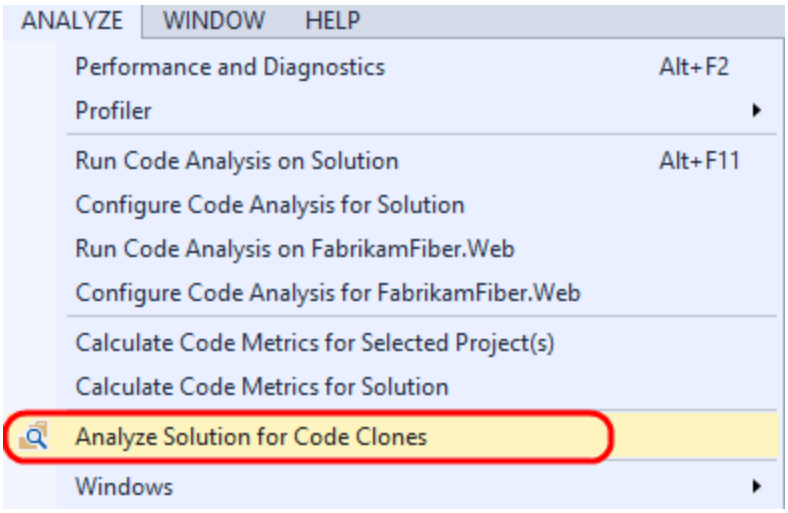
Изображение 46

Сохранение результатов анализа в файл

Упражнение 3: анализ похожего кода

В этом упражнении вы узнаете о том, как использовать представленную в Visual Studio 2012 функцию анализа похожего кода. Это средство предназначено для эвристического поиска семантически-похожего кода вместо поиска идентичного кода.

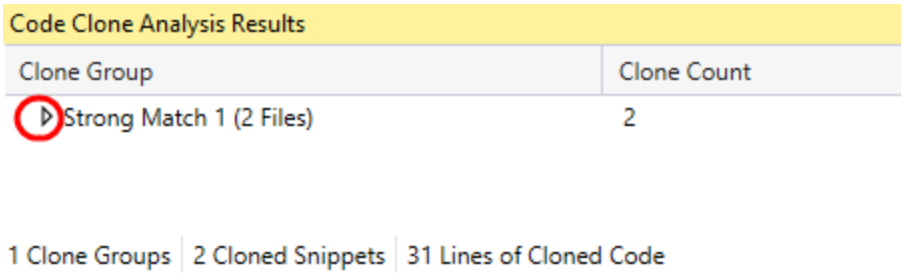
1. Нажмите на **Analyze | Analyze Solution for Code Clones**.



Изображение 47

Анализ похожего кода в решении

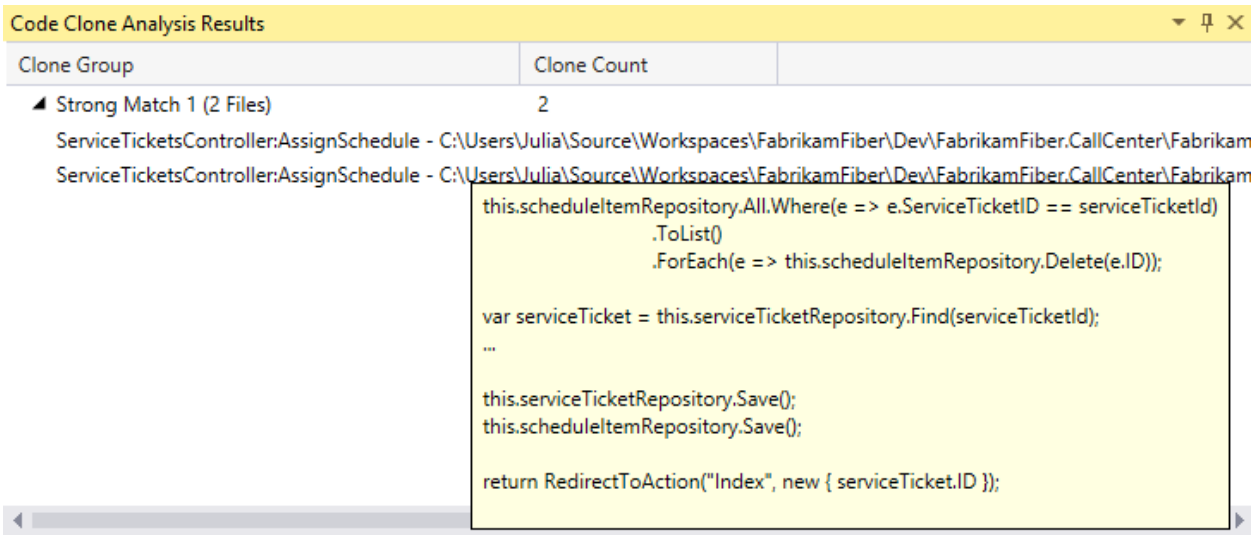
2. После анализа появится окно **Code Clone Analysis Results**, содержащее кандидатов на похожесть и сгруппированное по степени похожести. Разверните группу **Strong Match** – в ней два файла, которые содержат код сильной похожести.



Изображение 48

Результат анализа похожего кода

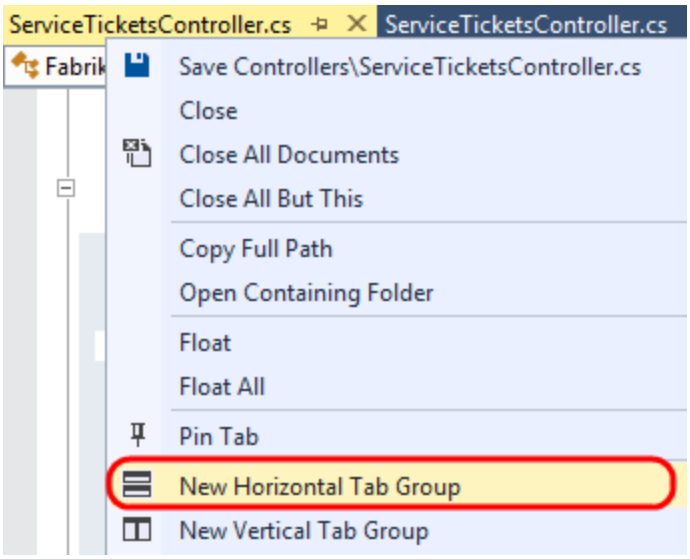
3. Каждая строка показывает класс и метод, файл и строку кода, в которой был найден похожий код. Наведение на строку показывает сниппет кода.



Изображение 49

Информация о результатах

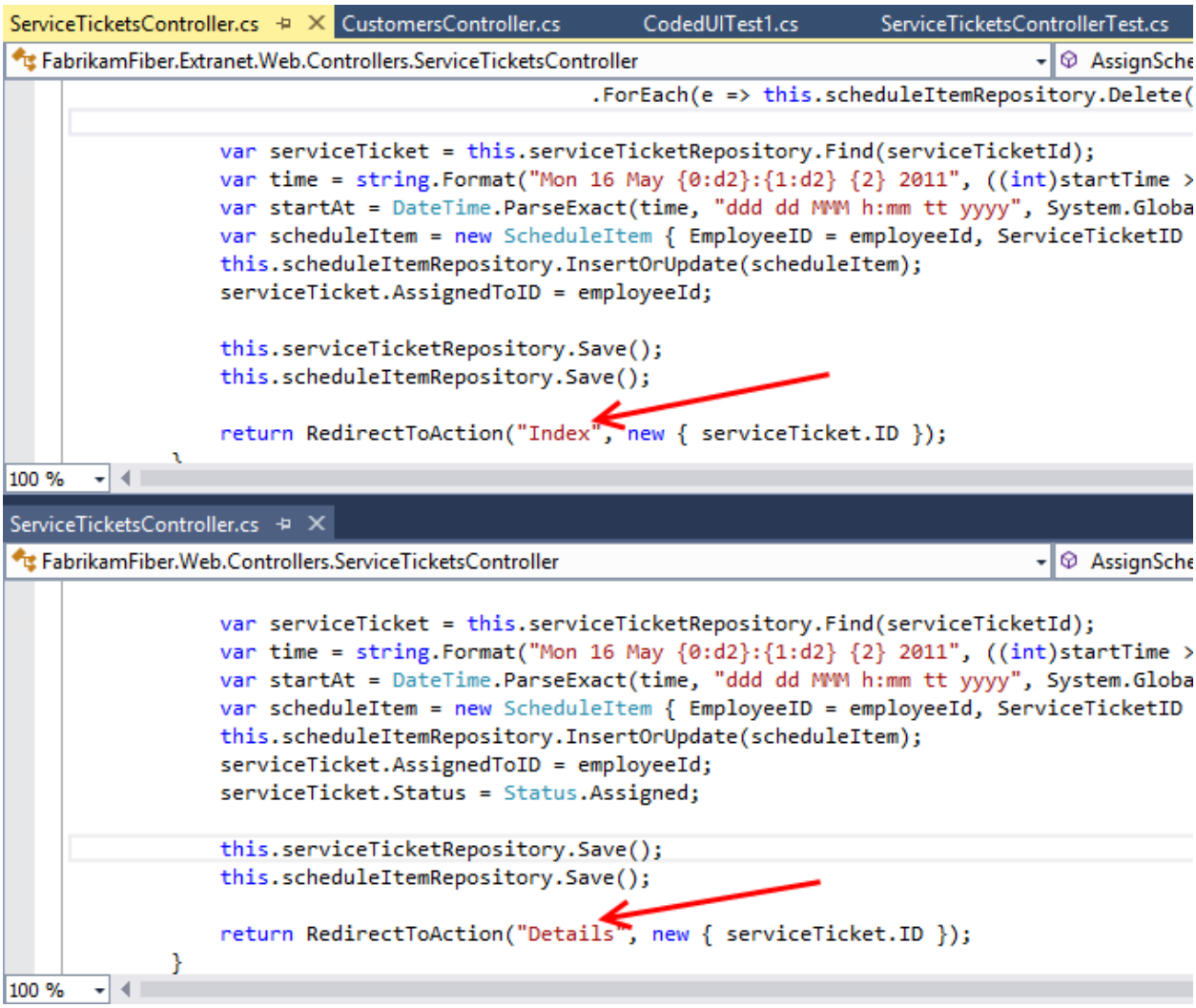
4. **Нажмите два раза** на каждой из строк, чтобы открыть код, **нажмите правой кнопкой** на названии первой, и выберите **New Horizontal Tab Group**.



Изображение 50

Сравнение двух файлов

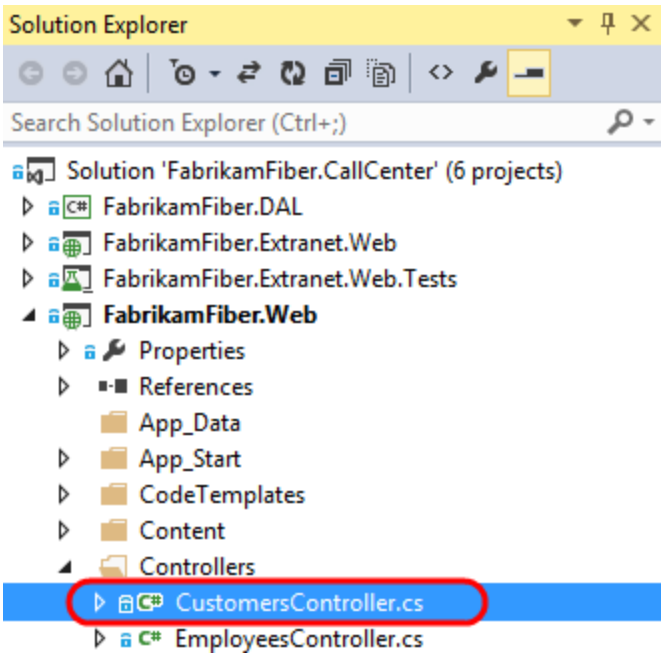
5. Найдите метод **AssignSchedule** в каждом из файлов. Оба блока кода идентичны за исключением последней строки, вызывающей метод **RedirectToAction**. Это значит, что этот метод – отличный кандидат на рефакторинг, и эта функция позволяет находить код, который можно оптимизировать.



Изображение 51

Сравнение результатов анализа похожего кода

6. Нажмите на **Window | Close All Documents**.
7. Можно сузить диапазон поиска. В **Solution Explorer** откройте файл **FabrikamFiber.Web | Controllers | CustomersController.cs**.



Изображение 52

Файл CustomersController.cs

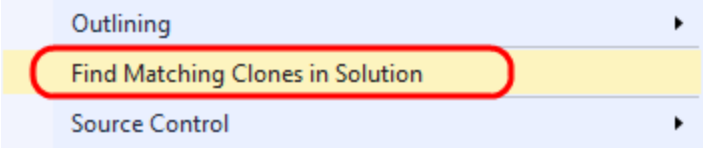
8. Найдите метод **Create,** принимающий параметр Customer, и выделите три строки кода внутри первого блока if.

```
[HttpPost]
2 references | 1/2 passing | Brian Keller +2 | 4 changes
public ActionResult Create(Customer customer)
{
    //check model state
    if (ModelState.IsValid)
    {
        this.customerRepository.InsertOrUpdate(customer);
        this.customerRepository.Save();
        return RedirectToAction("Index");
    }

    return this.View();
}
```

Изображение 53

Выбор кода для анализа

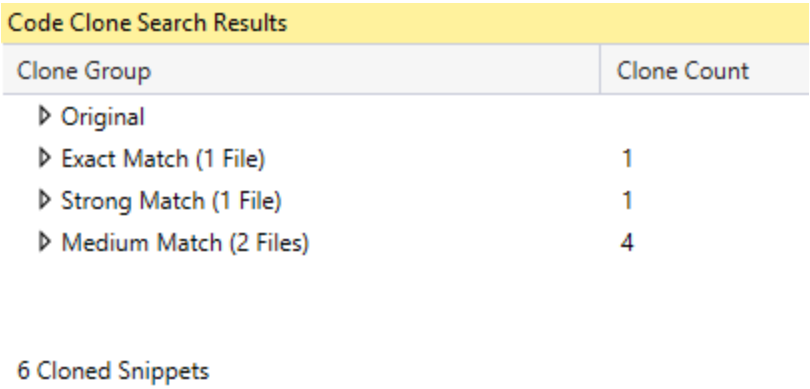
9. **Нажмите правой кнопкой** на выделенных строках и нажмите на **Find Matching Clones**.



Изображение 54

Поиск похожего кода

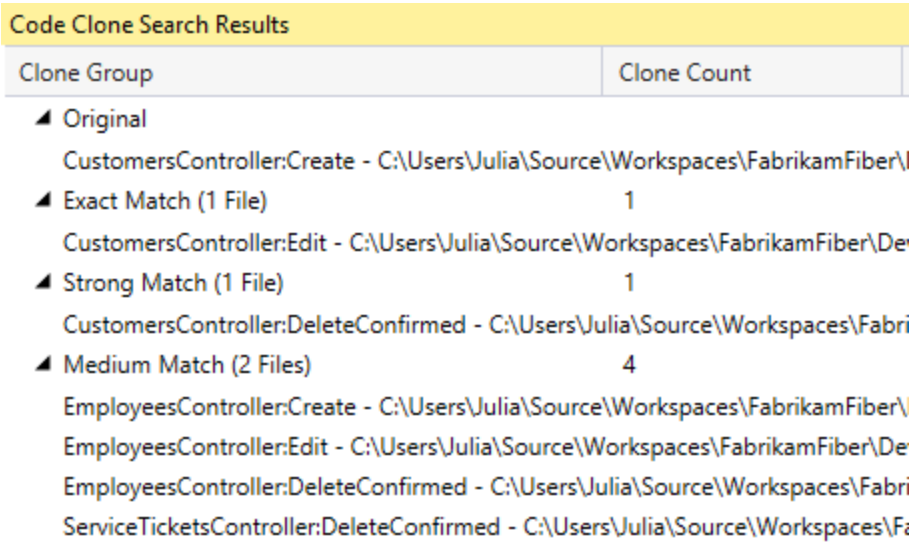
10. После поиска появится окно **Code Clone Search Results** с результатами анализа с различной степенью похожести.



Изображение 55

Результаты поиска похожего кода

11. **Разверните** все группы результатов.



Изображение 56

Результаты анализа

12. Наведите курсор на **Exact Match** – метод Edit использует идентичный методу Create код.
13. Наведите курсор на **Strong Match** – единственная разница – в первой строке с вызовом метода Delete.

```
this.customerRepository.Delete(id);
this.customerRepository.Save();

return RedirectToAction("Index");
```

Изображение 57
Сниппет похожего кода

14. Наведите курсор на первый результат **Medium Match** – этот блок кода похож на исходный, но мы работаем над совершенно другим объектом (сейчас **employeeRepository**).

```
this.employeeRepository.InsertOrUpdate(employee);
this.employeeRepository.Save();
return RedirectToAction("Index");
```

Изображение 58
Сниппет похожего кода

15. В целом, есть три сценария, в которых выявления похожего кода может быть полезным:
    a. Выявление кандидатов на рефакторинг
    b. Исправление ошибок и усовершенствование кода в процессе выявления разных блоков кода, которые нужно обновить
    c. Обучение нового разработчика – например, если разработчик добавляет код и хочет увидеть то, что уже было сделано в другом коде, например, использование блока Try-Catch.