МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет науки и технологий
имени академика М.Ф. Решетнева»

Кафедра информационных экономических систем

Рекомендовано
для использования в учебном процессе
методической комиссией
инженерно-экономического института
протокол № от «___»_____2017г.

Антамошкин Олеслав Александрович

Разработка и стандартизация программных средств и информационных технологий

Лабораторный практикум

для студентов направления 09.03.03 Прикладная информатика
образовательной программы «Прикладная информатика в экономике»

всех форм обучения

Красноярск, 2017

Общие рекомендации

Эта часть издания предназначена для практического усвоения дисциплины и использует в качестве инструментария продукт Microsoft Visual Studio Team System, поддерживающий многие практики командной разработки ПО. Освоение планирования, конфигурационного управления (средства контроля версий и управление сборками), автоматического тестирования и командной работы должно быть реализовано в рамках формально определенного процесса разработки.

Особо отметим, что нашей задачей является не столько изучение продукта MS VSTS, сколько использование его в качестве основы для практического освоения программной инженерии. В современном производстве без подобных инструментов уже не работают, и в принципе для обучения годится любой продукт такого класса. Достоинство продукта MS VSTS заключается в его доступности для целей обучения. Компания Microsoft ведет широкую работу по бесплатному распространению своих продуктов в образовательных целях в российских университетах, поэтому не составляет труда получить и данный продукт.

Модельная задача. Данный практикум проводится на основе практической задачи, которую Scrum-команда реализует в рамках практикума. При этом задача должны отвечать следующим требованиям:

- быть не очень трудной и допускать реализацию за 6–8 ч;
- быть распределенной (разные ее части, розданные разным участникам, должны быть зависимы друг от друга);
- иметь более широкий контекст, чтобы ее можно было выделить из более объемлющего backlog;
- хорошо подходить для написания модульных тестов;
- не содержать сложного пользовательского интерфейса или других технических сложностей.

В качестве варианта предлагается реализация игры «Балда» на компьютере. Правила игры можно найти в приложении.

Команде необходимо сформулировать задачу таким образом, чтобы разработать не столько отдельный графический интерфейс пользователя, сколько библиотеку классов, позволяющую легко реализовывать различные вариации этой игры с разным пользовательским интерфейсом или без него. Кроме того, одной из задач, стоящих перед командой, должна быть реализация компьютерного алгоритма этой игры.

Необходимо составить список пользовательских историй, описывающих необходимую функциональность задачи, должен быть достаточно детальным, чтобы максимально точно определить структуру

будущей системы и направить работу в нужное русло.

Практикум состоит из пяти тем. В зависимости от подготовленности участников может также потребоваться дополнительная информация, дополнительные детали по тем или иным основам MS VSTS, в частности по модульному тестированию.

Тема 1. При изучении первой темы участники организуются как Scrum-команда. Они также знакомятся с условиями модельной задачи, настраивают инфраструктуру TFS для будущей разработки (создают командный проект и распределяют права на работу с ним).

Тема 2. Обучающиеся практикуются в планировании работ на основе методологии Scrum, а также изучают способы использования системы отслеживания задач TFS. Необходимо импортировать список пользовательских историй их файла Excel в TFS, а затем детально спланировать будущий спринт и распределить задачи.

Тема 3. На этих занятиях осуществляется основная работа по реализации решения модельной задачи. Необходимо также освоиться с системой контроля версий TFS, ее интеграцией с системой отслеживания задач, а также попрактиковаться в создании ветвей и интеграции изменений.

Тема 4. Студенты практикуются в разработке модульных тестов средствами Visual Studio Team Developer. Они должны освоить средства автоматической генерации тестов, наполнить сгенерированные тесты содержимым, а также научиться изменять конфигурацию запуска модульных тестов и считать тестовое покрытие.

Тема 5. Занятия посвящены системе автоматических сборок TFS. Участники должны создать несколько определений для автоматической сборки (build definitions) в разных случаях – с тестами и без, с анализом кода и без и т.д., настроить параметры непрерывной интеграции и рассылки уведомлений.

Тема 6. Студенты должны провести ретроспективный анализ выполненного Scrum sprint, выявить потенциальные способы оптимизации, а затем применить их, используя средства настройки процесса разработки TFS. Также студенты должны освоить изменение настроек системы остлеживания задач средствами Team Foundation Server Power Tools.

ТЕМА 1. ОЗНАКОМЛЕНИЕ С ЗАДАЧЕЙ И СОЗДАНИЕ ПРОЕКТА

Цели занятия:

1. Разделить студентов на Scrum-команды, определить и обсудить Scrum-роли.
2. Провести начальное ознакомление с модельной задачей, которую предстоит реализовать.

3. Прояснить открытые вопросы по модельной задаче с Prodcut Owner.

4. Создать командный проект на TFS и добавить в него пользователей.

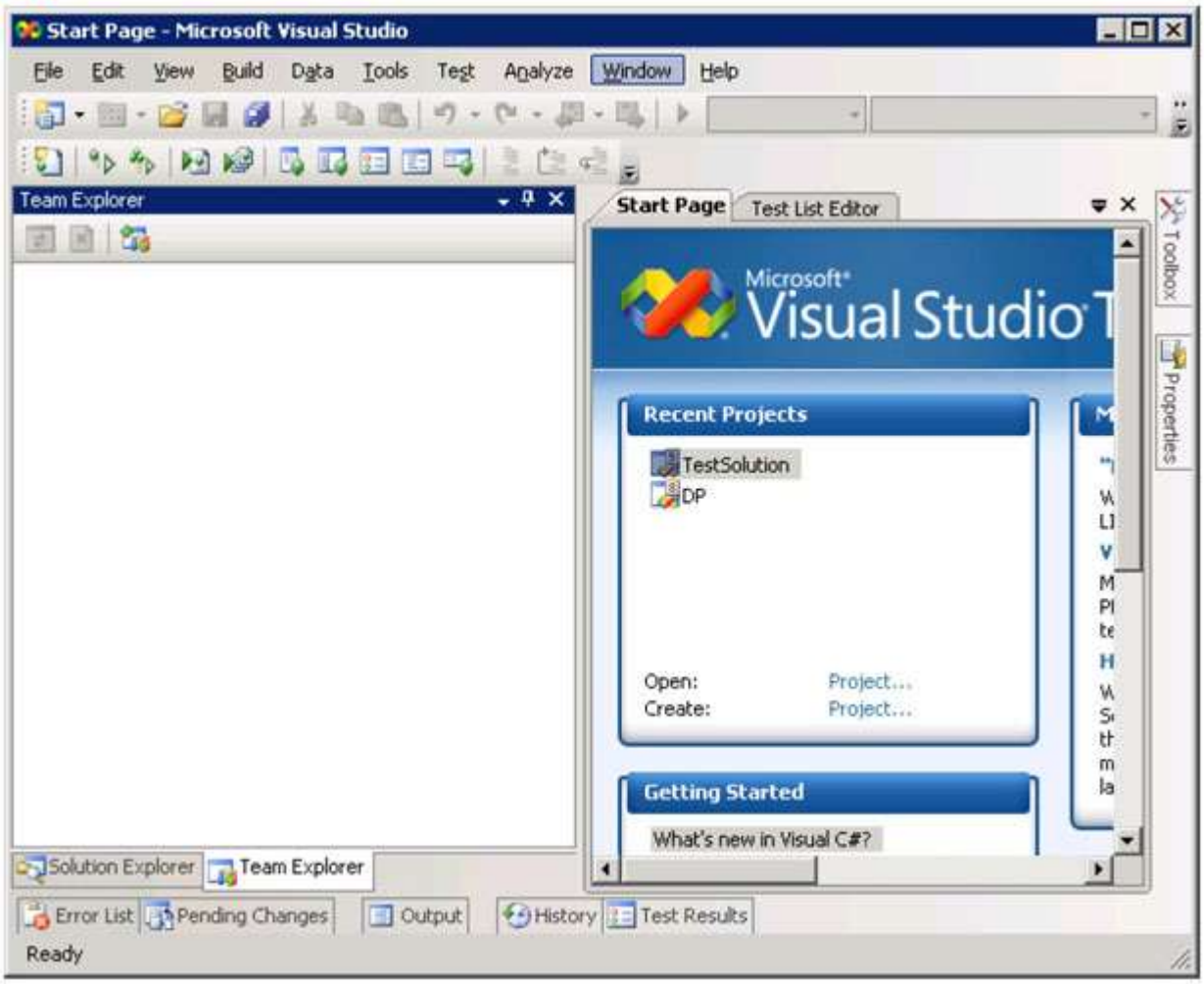
Для разделения на Scrum-команды и выделения Scrum-мастеров можно прибегнуть к жеребьевке, после чего перейти к знакомству с задачей. Необходимо заранее приготовить краткое описание основной концепции разрабатываемого приложения и список задач в формате product backlog.

Команды в течение 20–30 мин. обсуждают их, готовя вопросы к хозяину продукта. Затем в течение 20–30 мин. хозяин продукта отвечает на подготовленные командами вопросы.

После того как команды получили представление о разрабатываемом продукте (модельной задаче), им необходимо создать командный проект в MS VSTS и занести всех участников проекта в список пользователей.

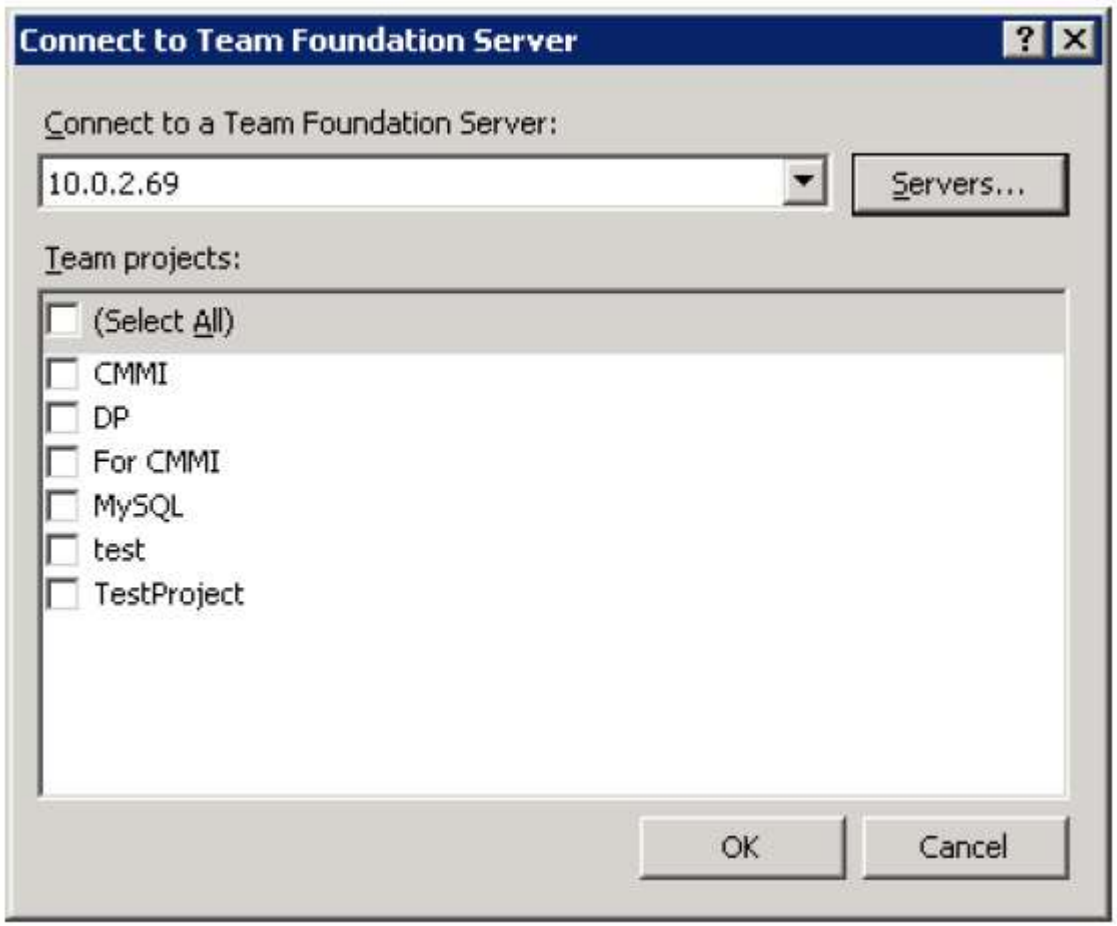
Шаг 1. Создание проекта. Командный проект создается лидером команды по следующему сценарию:

1. Открыть Visual Studio Team Suite и окно Team Explorer в ней:



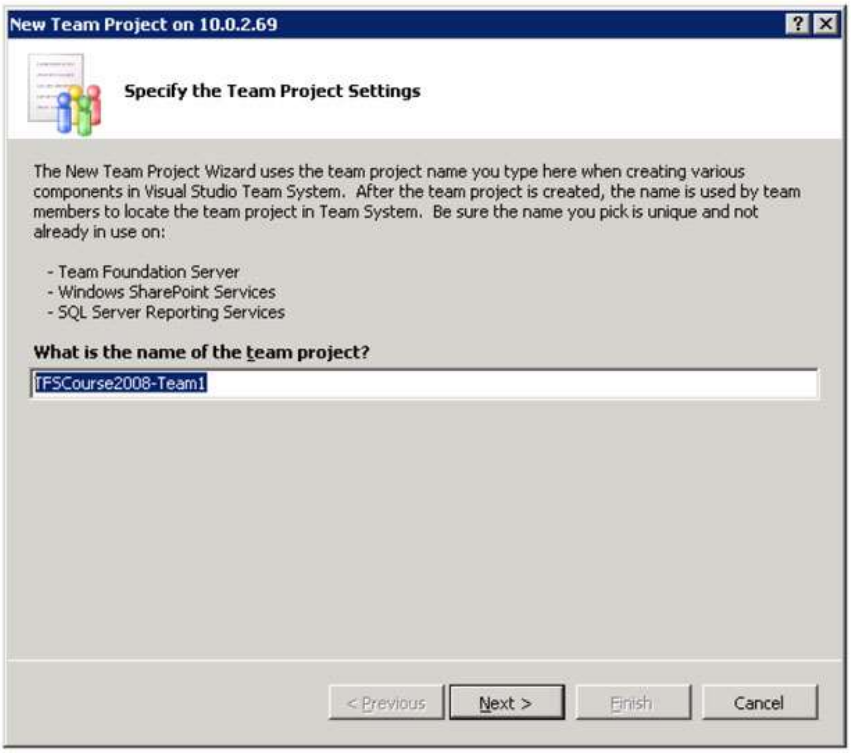
2. Нажать кнопку «Соединиться с сервером»  .

3. Задать имя и порт сервера в открывшемся диалоге:

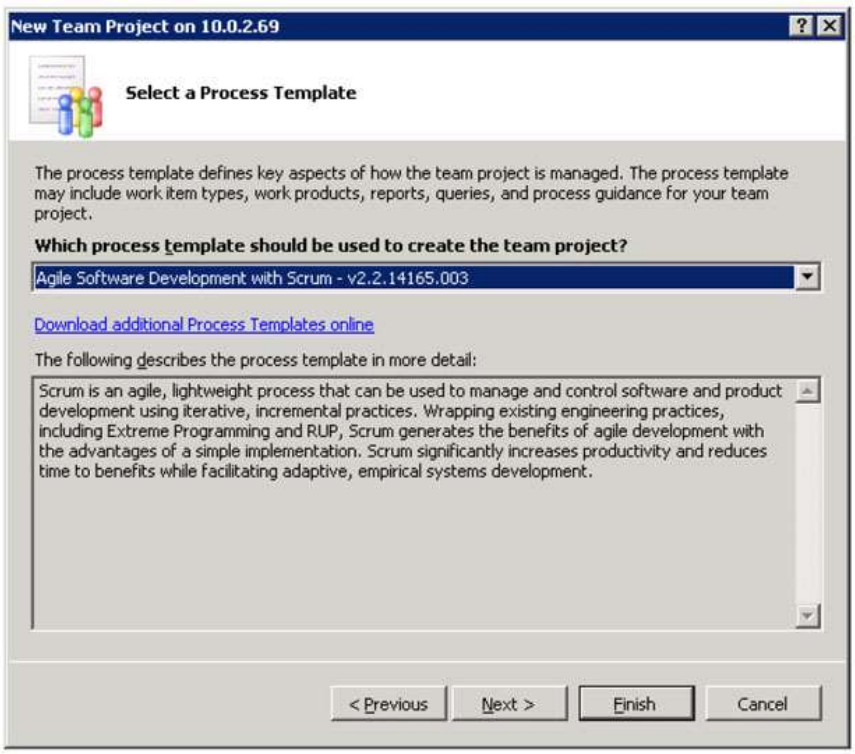


4. После того как соединение с сервером установлено, запустить процедуру создания проекта, используя соответствующую команду контекстного меню (New Team Project…):

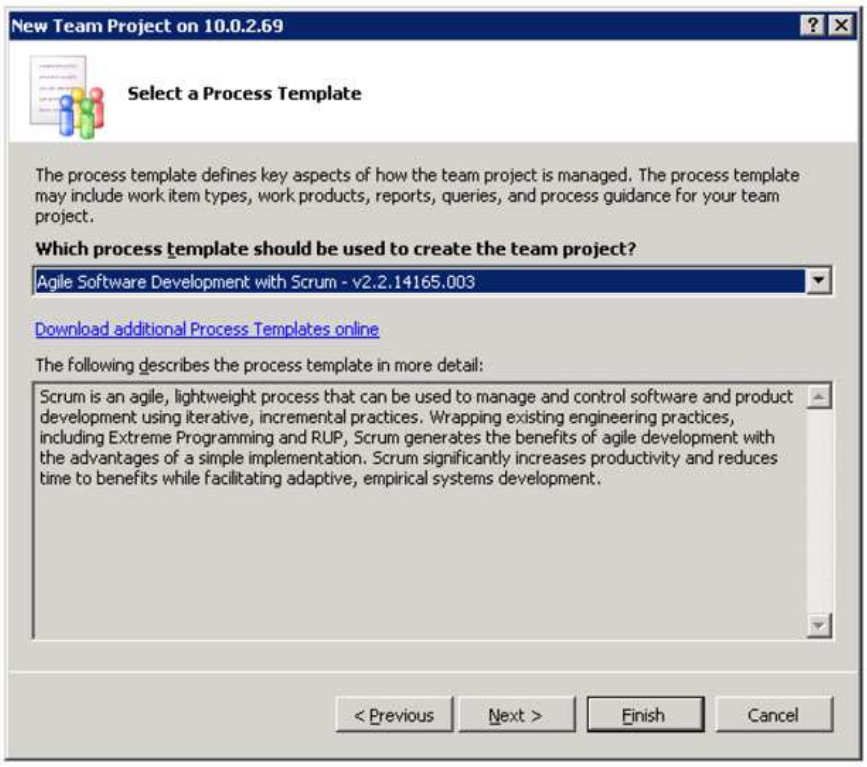
5. В качестве имени проекта необходимо указать TFSCourse<год>-<имя команды>:



6. В качестве шаблона процесса разработки выбрать Scrum:



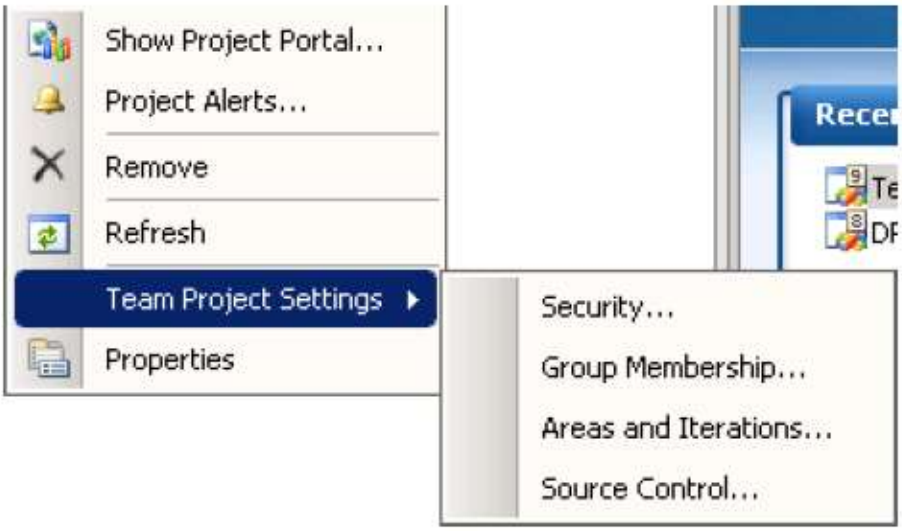
7. Создать новый пустой раздел в системе контроля версий для данного проекта:



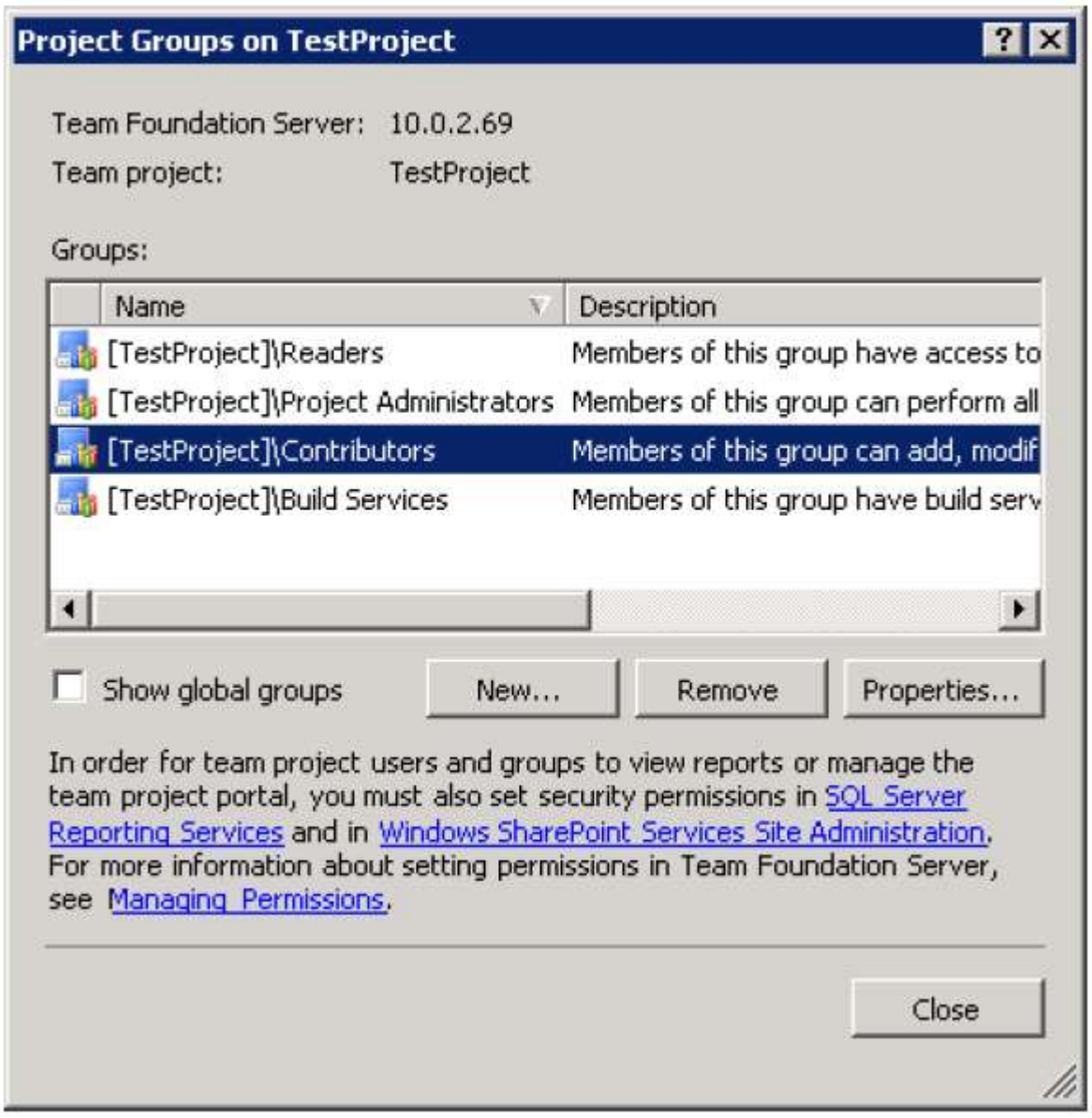
8. После выбора всех настроек создать проект.

Шаг 2. Настройка прав. После того как проект был создан, лидеру необходимо выделить права остальным участникам команды для работы с этим проектом. Для этого ему нужно:

1. Выбрать в свойствах проекта раздел Group membership:



2. В открывшемся диалоге включить всех участников в группы Readers и Contributors:



Шаг 3. Подключение проекта остальными участниками команды. После того как все получили права на работу с проектом, каждый участник команды должен на своей машине открыть Visual Studio и добавить соединение с этим проектом. Выполняется это аналогично пунктам 1–4 первого шага занятия.

ТЕМА 2. РАБОТА С СИСТЕМОЙ ОТСЛЕЖИВАНИЯ ОШИБОК

Цель занятия – ознакомление участников с системой отслеживания элементов работы, а именно:

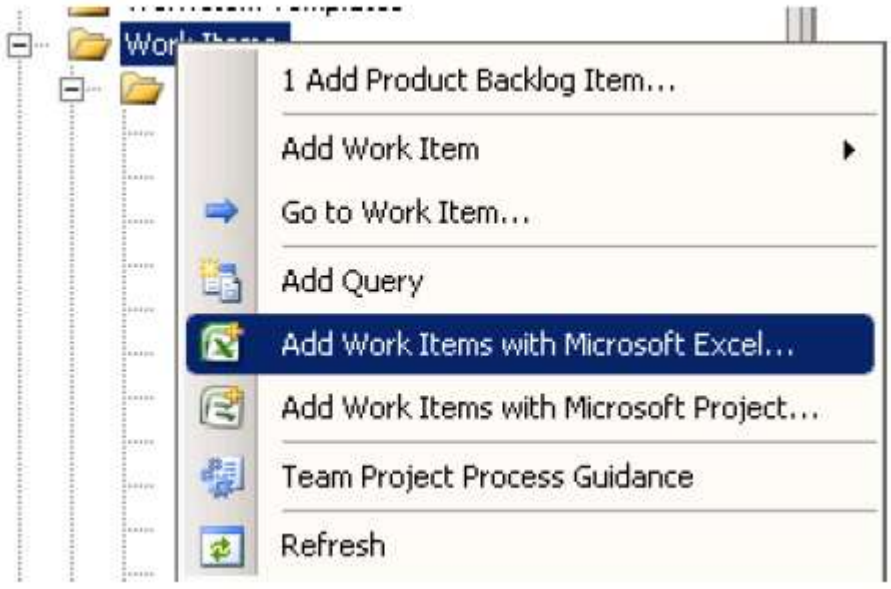
- создание элементов работы средствами Visual Studio и Team Explorer;
- импорт и экспорт элементов работы из/в Mcrosoft Excel;
- назначение ответственных за элементы работы;
- отслеживание текущего статуса посредством отчетов.

В рамках данного занятия необходимо осуществить планирование работы команды по методологии Scrum. Команды уже предварительно ознакомились с проектом, а на этом этапе от них требуется:

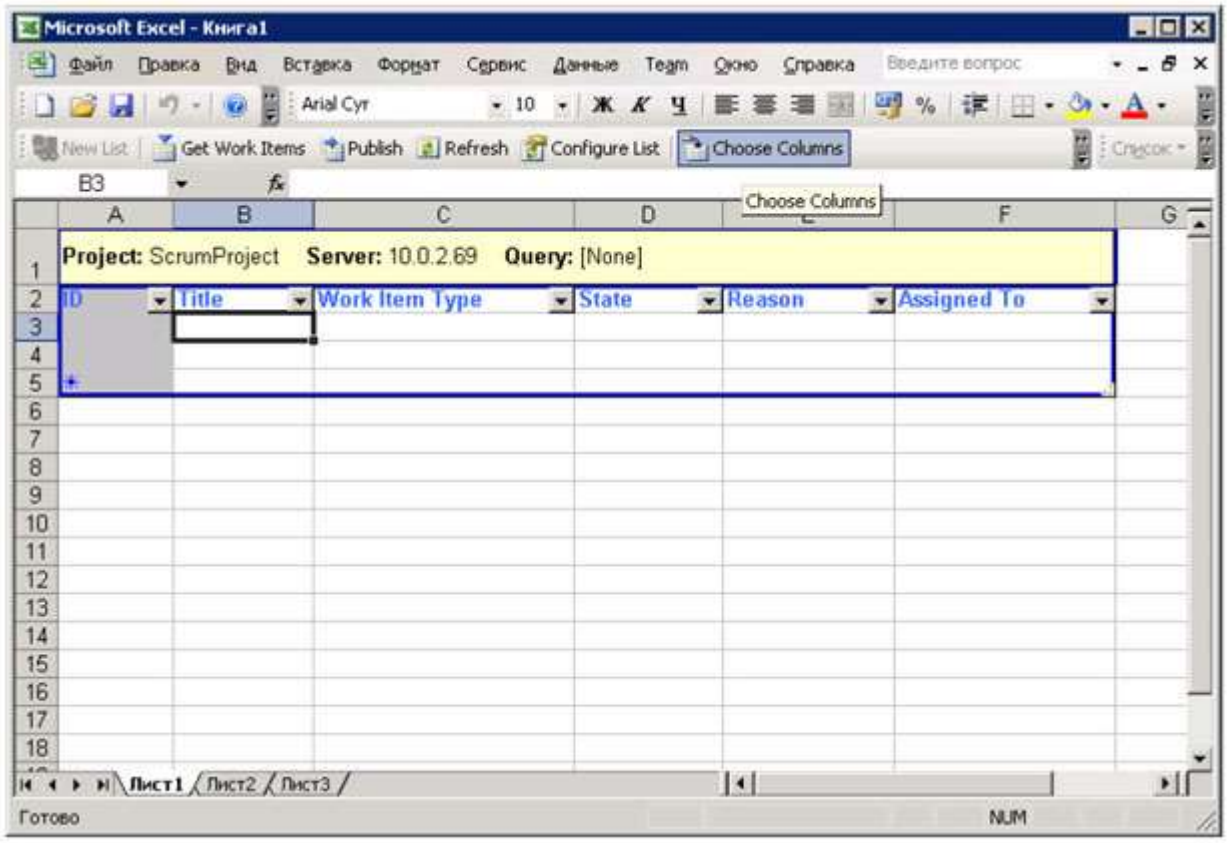
1. Импортировать содержимое списка требований в TFS, используя средства импорта из Excel.
2. Подробно рассмотреть 10 наиболее приоритетных пользовательских историй.
3. Обсудить возникшие вопросы с хозяином продукта.
4. Провести детальное планирование и разбить пользовательские истории на мелкие подзадачи.
5. Распределить подзадачи среди участников проекта.
6. Отчитаться перед хозяином продукта о том, какие пользовательские истории были запланированы. При отчете использовать отчеты TFS.

Шаг 1. Импорт списка пользовательских историй. Для того чтобы загрузить пользовательские истории из Excel-файла, полученного командами на прошлом занятии, нужно:

1. Выбрать команды Add Work Items with Microsoft Excel в контекстном меню:



2. В открывшемся окне Excel настроить колонки таким образом, чтобы они совпадали по расположению и смыслу с колонками в исходном Excel–документе (для этого можно использовать команды Choose Columns):

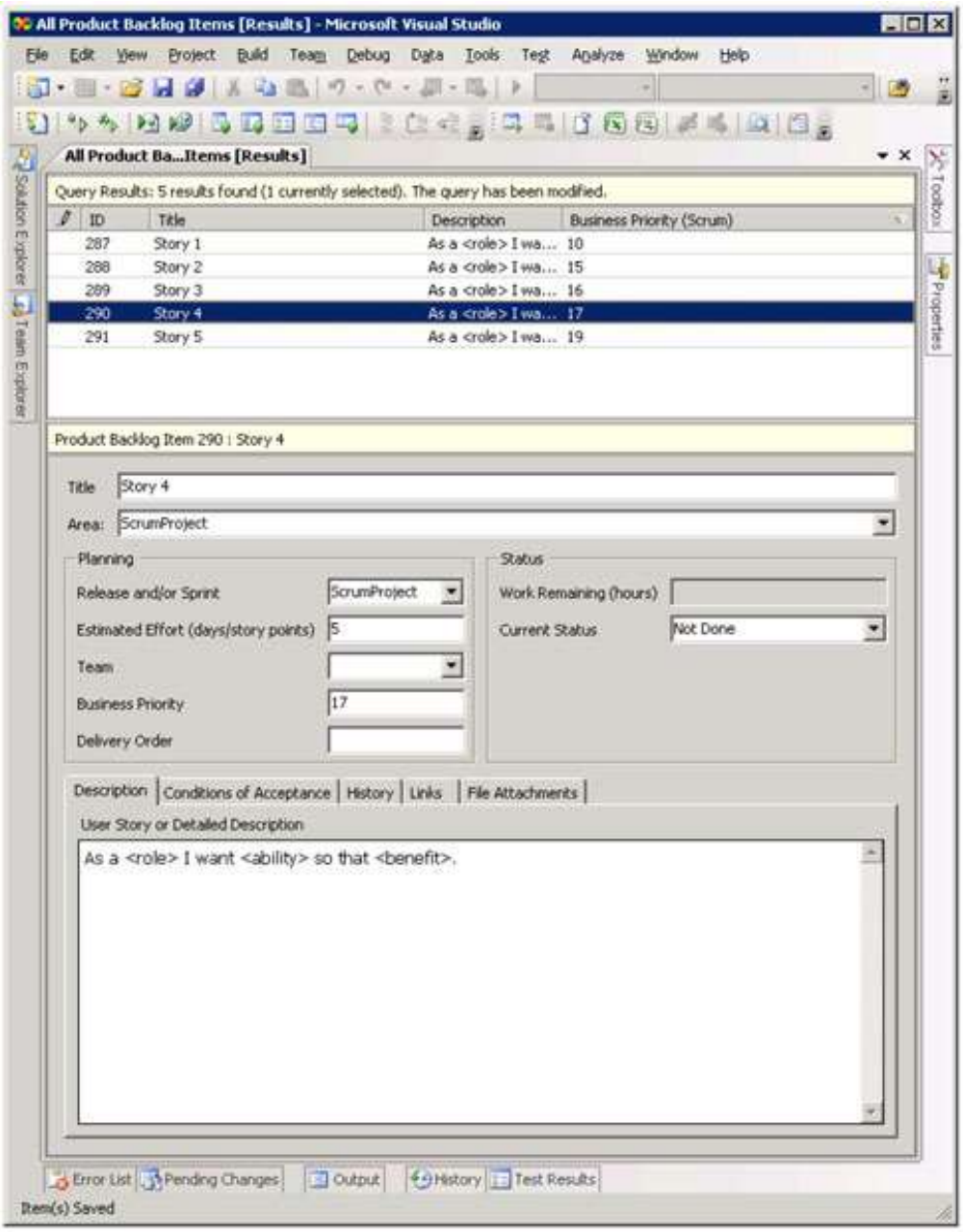


3. Скопировать значения из исходного Excel–документа в редактируемый.

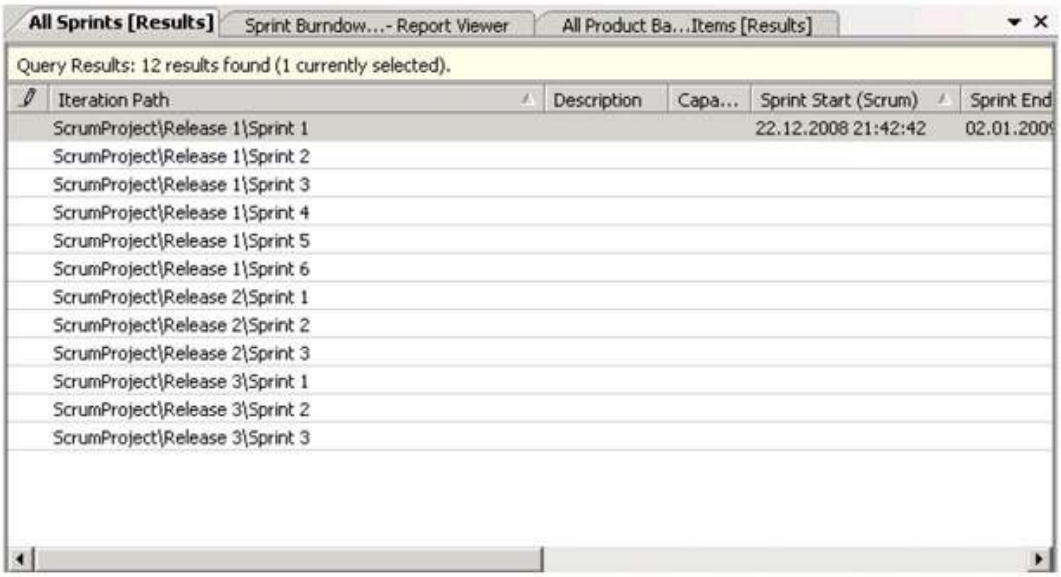
4. Показать колонку с именем Work Item Type и задать для всех строчек значение Product Backlog Item.

5. Нажать кнопку 

6. Убедиться, что при выполнении запроса All Product Backlog Items, видны все вновь загруженные пользовательские истории:



Шаг 2. Создание sprint. После импорта списка пользовательских историй команда должна создать элемент работы, соответствующий начинающемуся sprint. Некоторое количество sprints уже создано по умолчанию при создании проекта:



Для активации первого спринта ему необходимо установить дату начала, дату окончания и количество часов, которые команда может потратить в этом спринте.

Шаг 3. Формирование Sprint backlog. После обсуждения открытых вопросов по 10 наиболее приоритетным пользовательским историям команда должна приступить к планированию текущего sprint и формированию sprint backlog. Для этого ей необходимо рассмотреть список всех пользовательских историй и разбить его на список более мелких задач. При этом для каждой задачи нужно создать элемент работы типа sprint backlog item и проставить следующие атрибуты:

1. В качестве sprint указать Release 1/Sprint 1.
2. Добавить связь с соответствующим элементом product backlog, а также со всеми связанными элементами работы.
3. Установить Estimated efforts и Work remaining в соответствии с оценкой команды.
4. Задать ответственного за задачу (Owned By).

Выполнить все операции нужно средствами Visual Studio и Team Explorer.

После создания и распределения задач каждый член команды должен на своей машине убедиться, что выданные ему задачи отображаются в результатах запроса My Sprint Backlog Items.

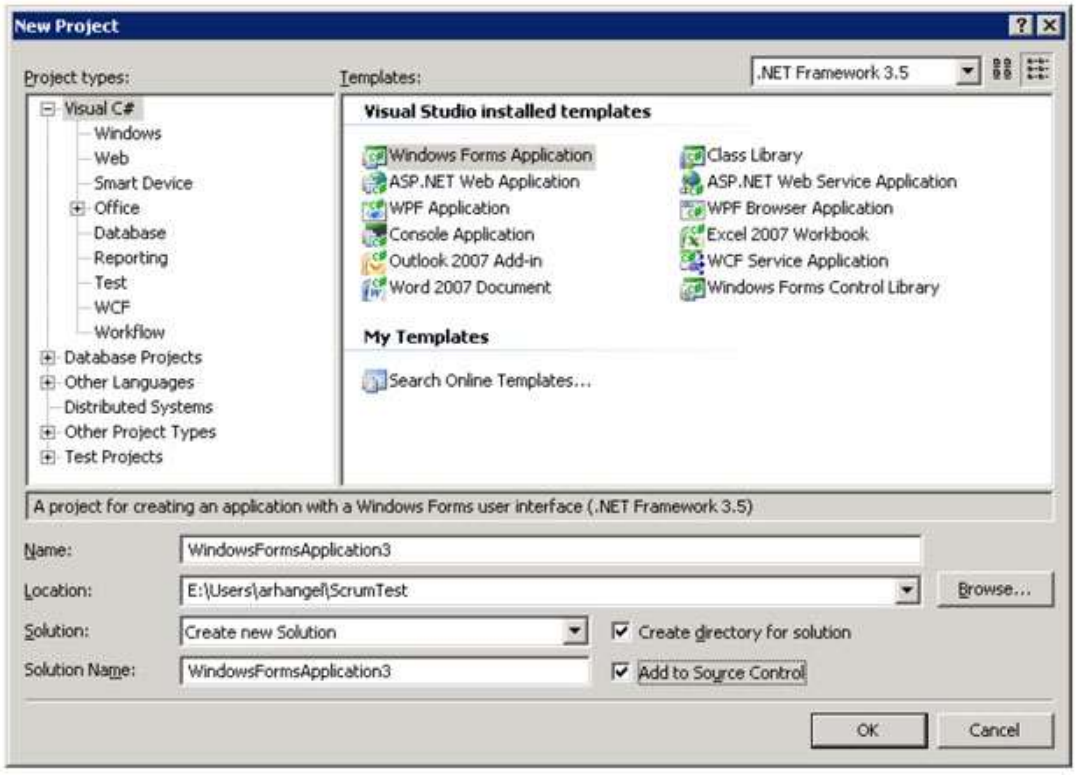
ТЕМА 3. РАБОТА С СИСТЕМОЙ КОНТРОЛЯ ВЕРСИЙ

Цель занятия – освоение системы контроля версий Team Foundation Server и ее интеграции с системой отслеживания задач.

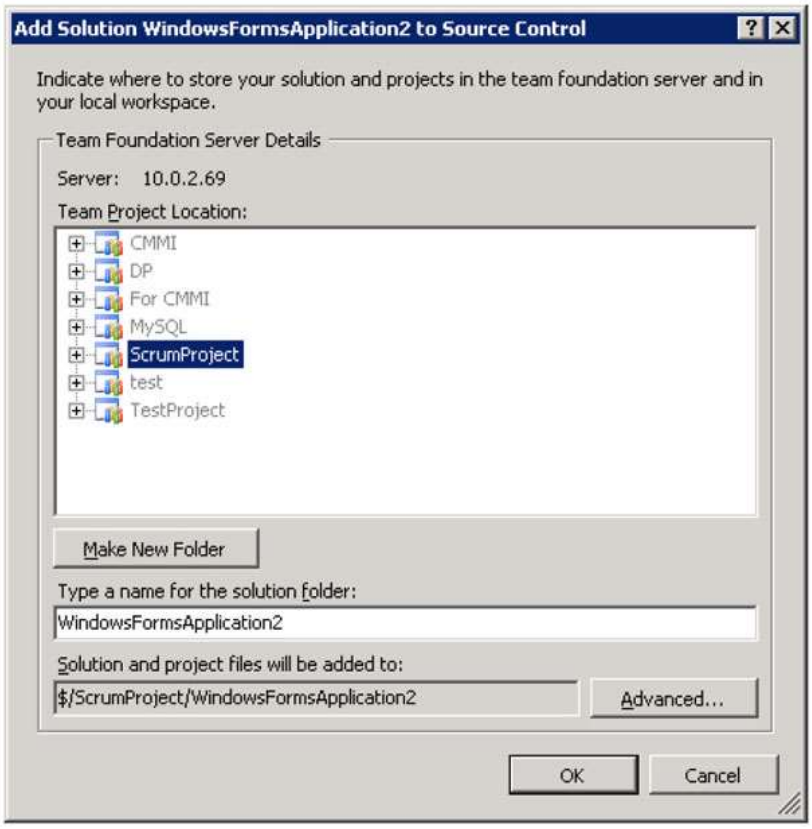
Занятие предполагает выполнение следующих действий:

1. Разработка кода модельной задачи средствами Visual Studio и внесение его в систему управления версиями.
2. Проставление связей между вносимыми изменениями и элементами системы отслеживания задач.
3. Создание параллельно поддерживаемых веток кода.
4. Интеграция изменений, сделанных параллельно в одном файле или в разных ветках кода.

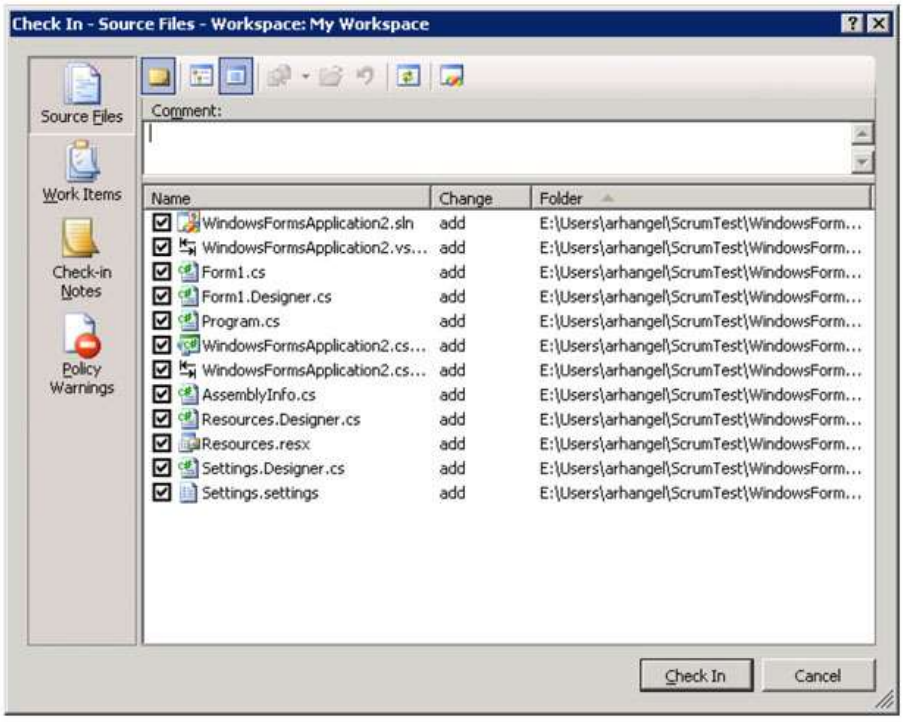
Шаг 1. Разработка кода. Перед началом работы команде необходимо создать решение (solution) средствами Visual Studio, включив опцию Add to Source Control:



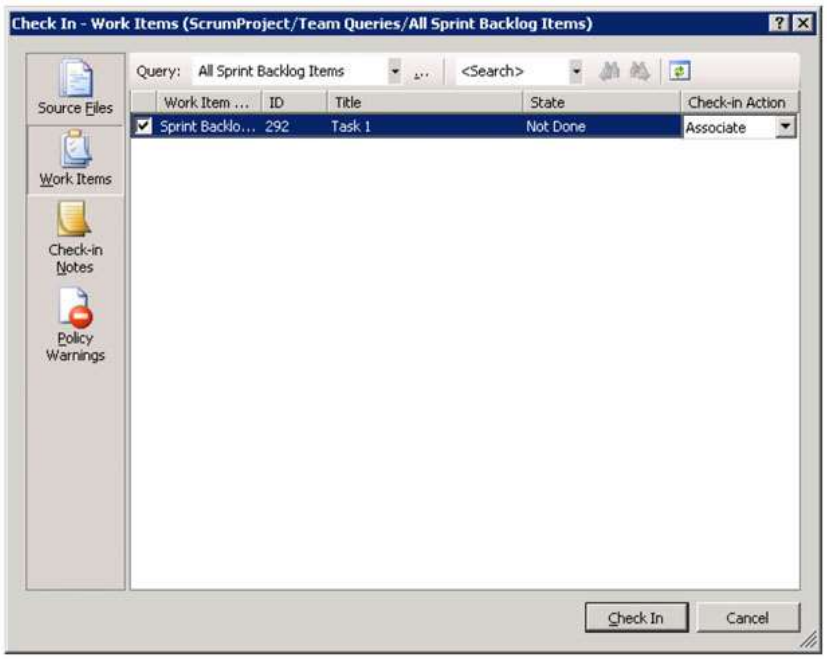
В открывшемся после создания проекта окне необходимо выбрать командный проект, в систему контроля версий которого нужно добавить данное решение:



Затем следует внести все данные в систему контроля версий, используя команду Check-in, открывающую диалог:

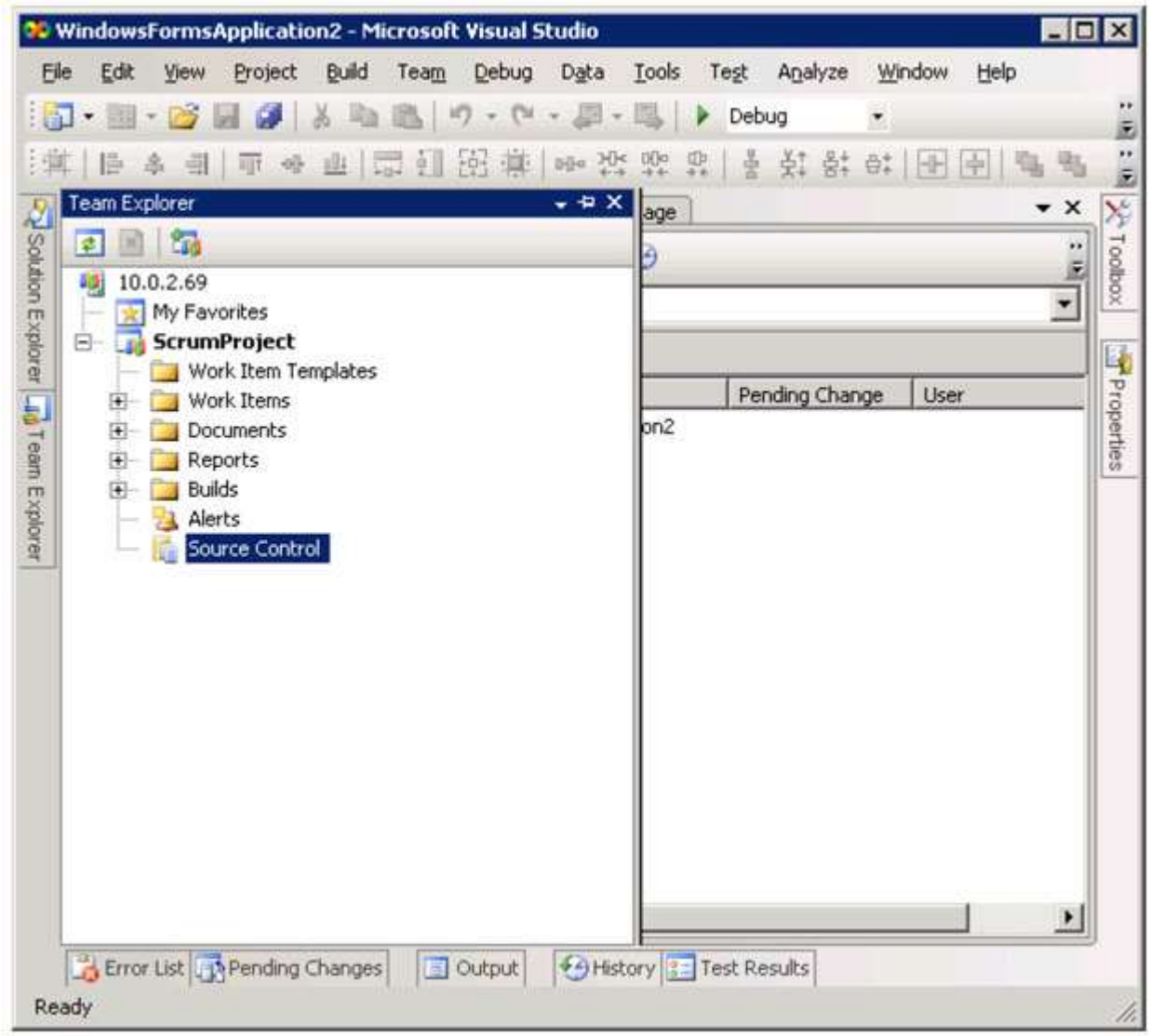


В этом диалоге необходимо ввести комментарии к вносимому коду, а также на вкладке Work Items связать вносимое изменение с элементами работы:

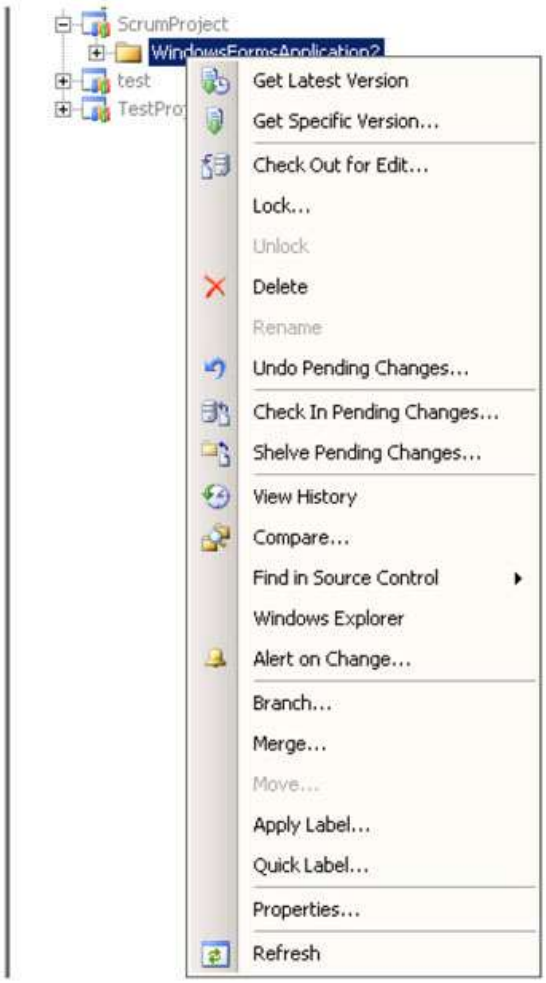


Шаг 2. Создание ветки кода. Для того чтобы освоиться с практикой конфигурационного управления, команды должны создать ветвь в системе контроля версий, следуя приведенной ниже инструкции.

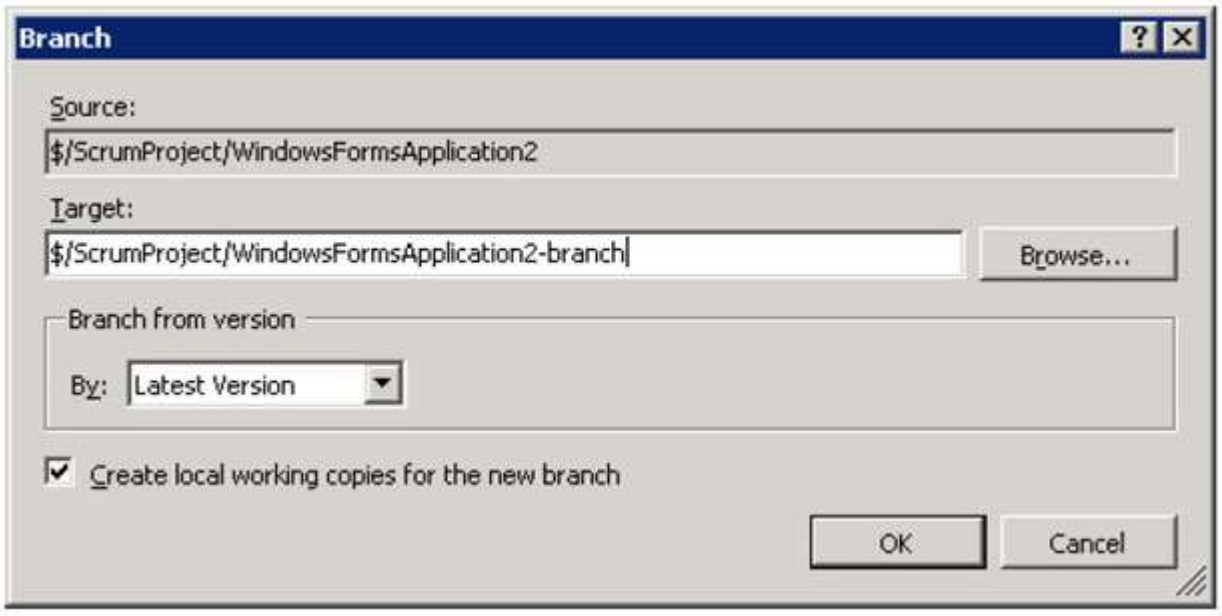
1. Открыть Source Control Explorer:



2. Выбрать нужный проект и в контекстном меню команду Branch:



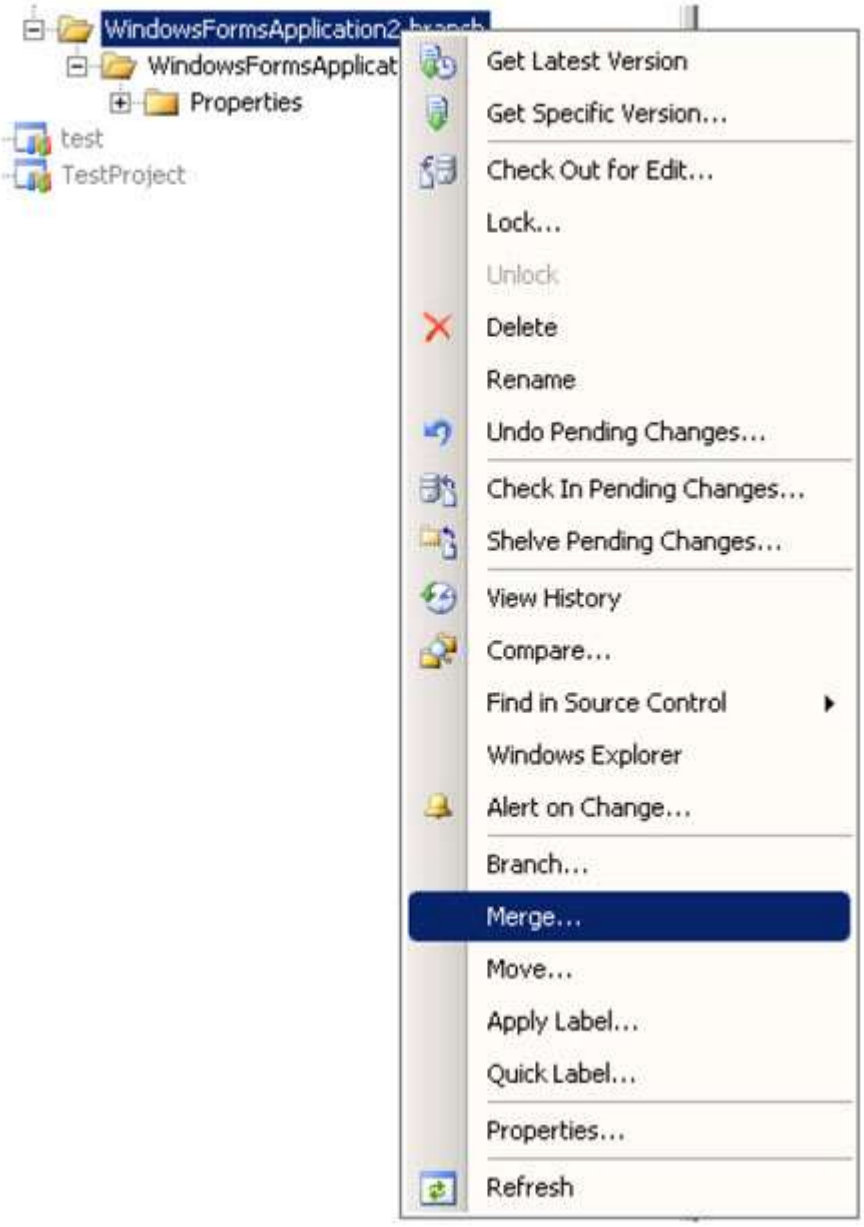
3. В открывшемся окне задать целевую папку, куда необходимо скопировать данные для новой ветви:



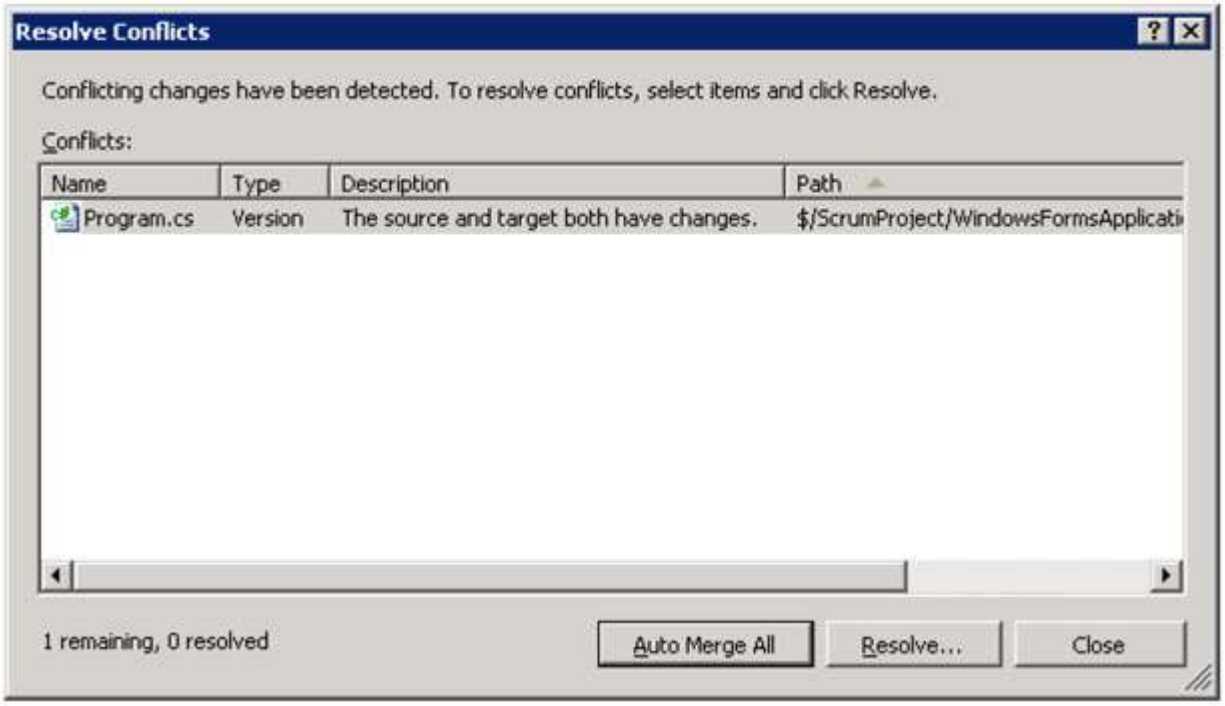
4. Внести изменения с помощью команды Check-in

После того как создана ветка, разные участники команды вносят изменения в разные ветки кода, реализуя необходимую функциональность приложения.

Шаг 3. Объединение изменений. После внесения отдельные ветви некоторого количества изменений необходимо перенести их из отдельной ветви в основную, используя команду Merge:



В процессе объединения изменений могут возникнуть конфликты, информация о которых будет включена в сообщение следующего вида:



Все конфликты необходимо разрешить, используя команду Resolve и утилиту для объединения результатов.

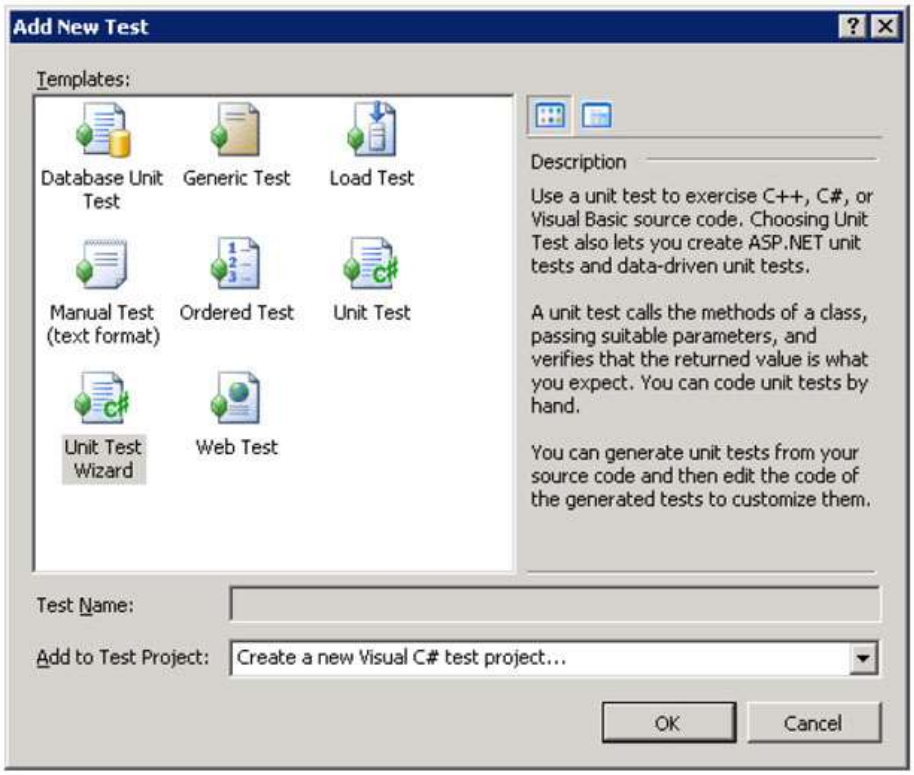
После разрешения конфликтов все изменения нужно внести в систему контроля версий посредством операции Check-in.

ТЕМА 4. РАЗРАБОТКА МОДУЛЬНЫХ ТЕСТОВ

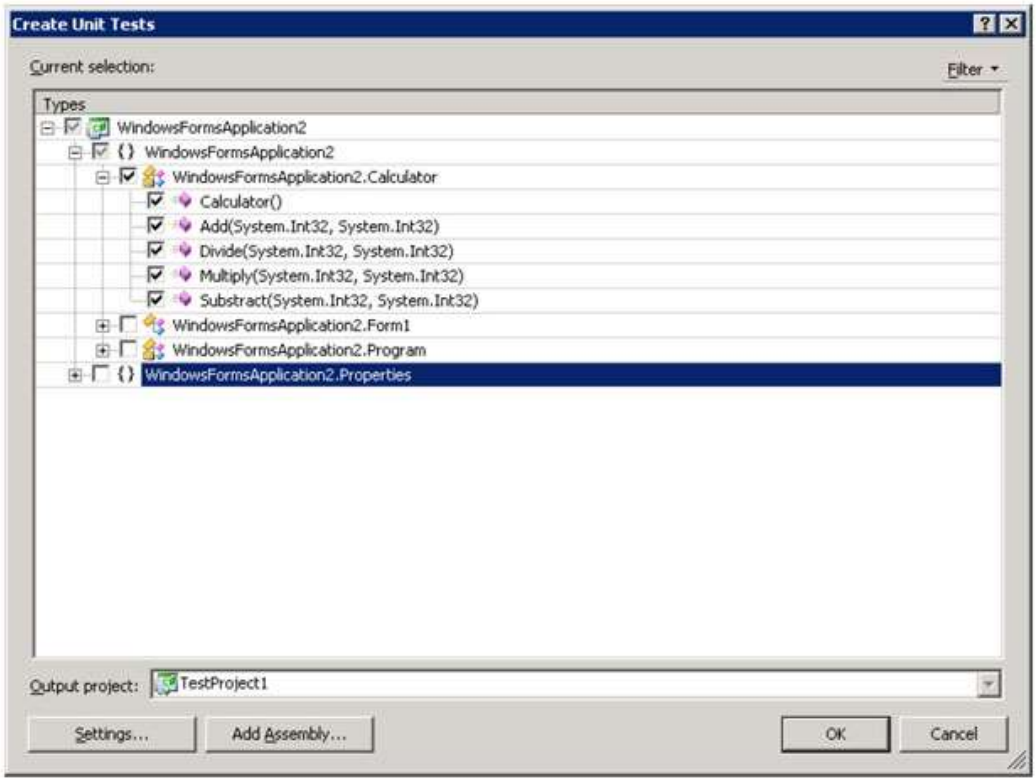
Цель занятия – создание командами набора модульных тестов, покрывающих функциональность, разработанную на предыдущем занятии. В рамках данного занятия предполагается освоить следующие возможности MS VSTS:

- автоматическую генерацию тестов;
- наполнение тестов содержимым;
- запуск тестов и просмотр результатов;
- изменение конфигурации работы тестов.

Шаг 1. Автоматическая генерация тестов. Для ускорения разработки тестов команды могут воспользоваться возможностью Visual Studio по автоматической генерации тестов. Для этого необходимо использовать команду Test/New Test и выбрать Unit Test wizard в открывшемся окне:



После создания тестового проекта будет предложен выбор из тех типов и методов, для тестирования которых необходимо разработать заглушки:



В этом диалоге команда должна выбрать все основные классы и методы, которые планируется покрыть модульными тестами.

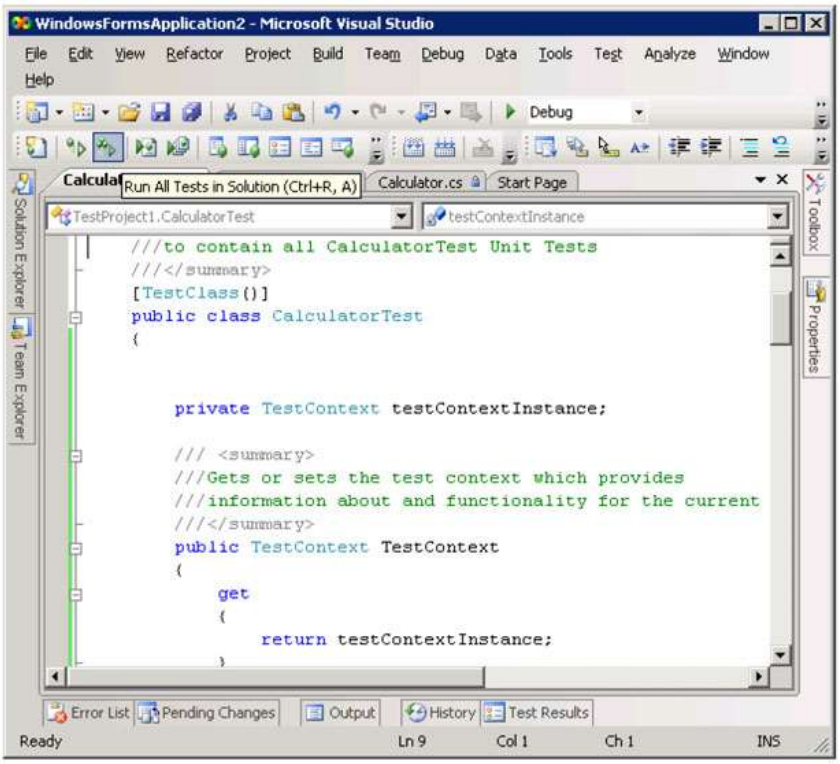
Шаг 2. Наполнение тестов содержимым. После генерации тестового покрытия команда должна получить набор скелетов тестов для всех методов, которые были выбраны для тестирования. Однако эти тесты имеют достаточно простую структуру и пока лишены смысла:

```
/// <summary>
///A test for Multiply
///</summary>
[TestMethod()]
public void MultiplyTest()
{
// TODO: Initialize to an appropriate value
Calculator target = new Calculator();
int a = 0; // TODO: Initialize to an appropriate value
int b = 0; // TODO: Initialize to an appropriate value
int expected = 0; // TODO: Initialize to an appropriate value
int actual;
int a = 0; // TODO: Initialize to an appropriate value
int b = 0; // TODO: Initialize to an appropriate value
int expected = 0; // TODO: Initialize to an appropriate value
```

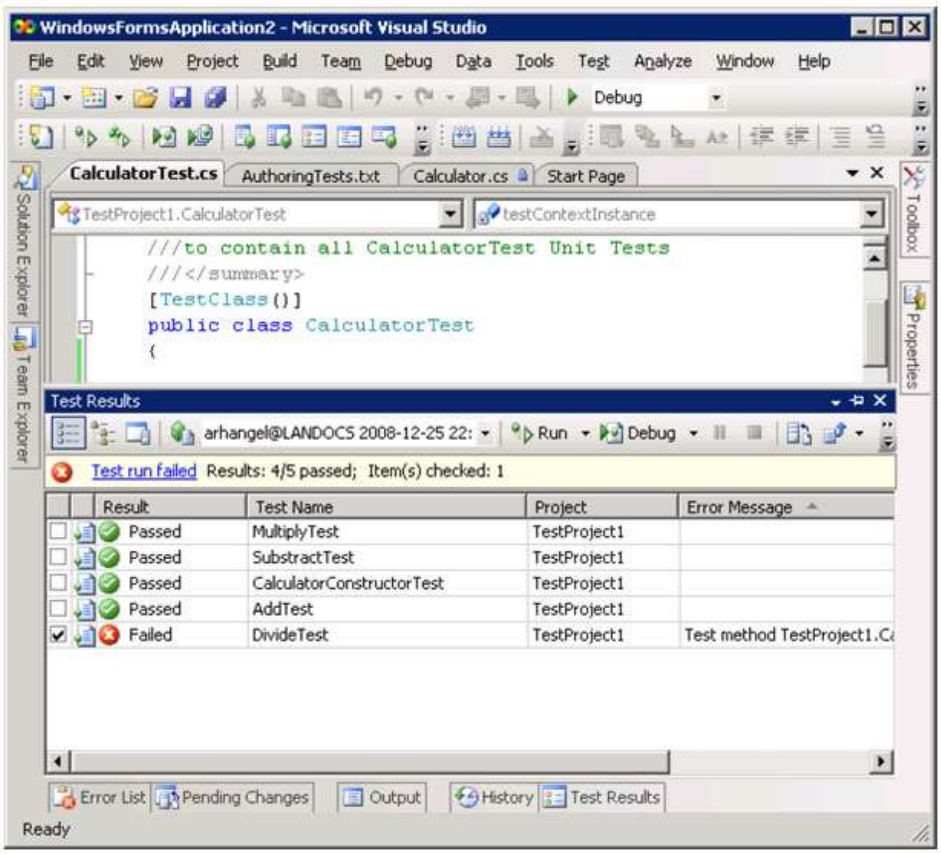
```
int actual;
actual = target.Multiply(a, b);
Assert.AreEqual(expected, actual);
Assert.Inconclusive("Verify the correctness of this test
method.");
}
```

На следующем шаге команда должна заполнить эти тесты необходимым содержимым, используя функциональность по валидации (Assert), предоставляемую тестовой платформой.

Шаг 3. Запуск тестов. Для того чтобы исполнить созданные тесты, необходимо использовать соответствующую панель инструментов:



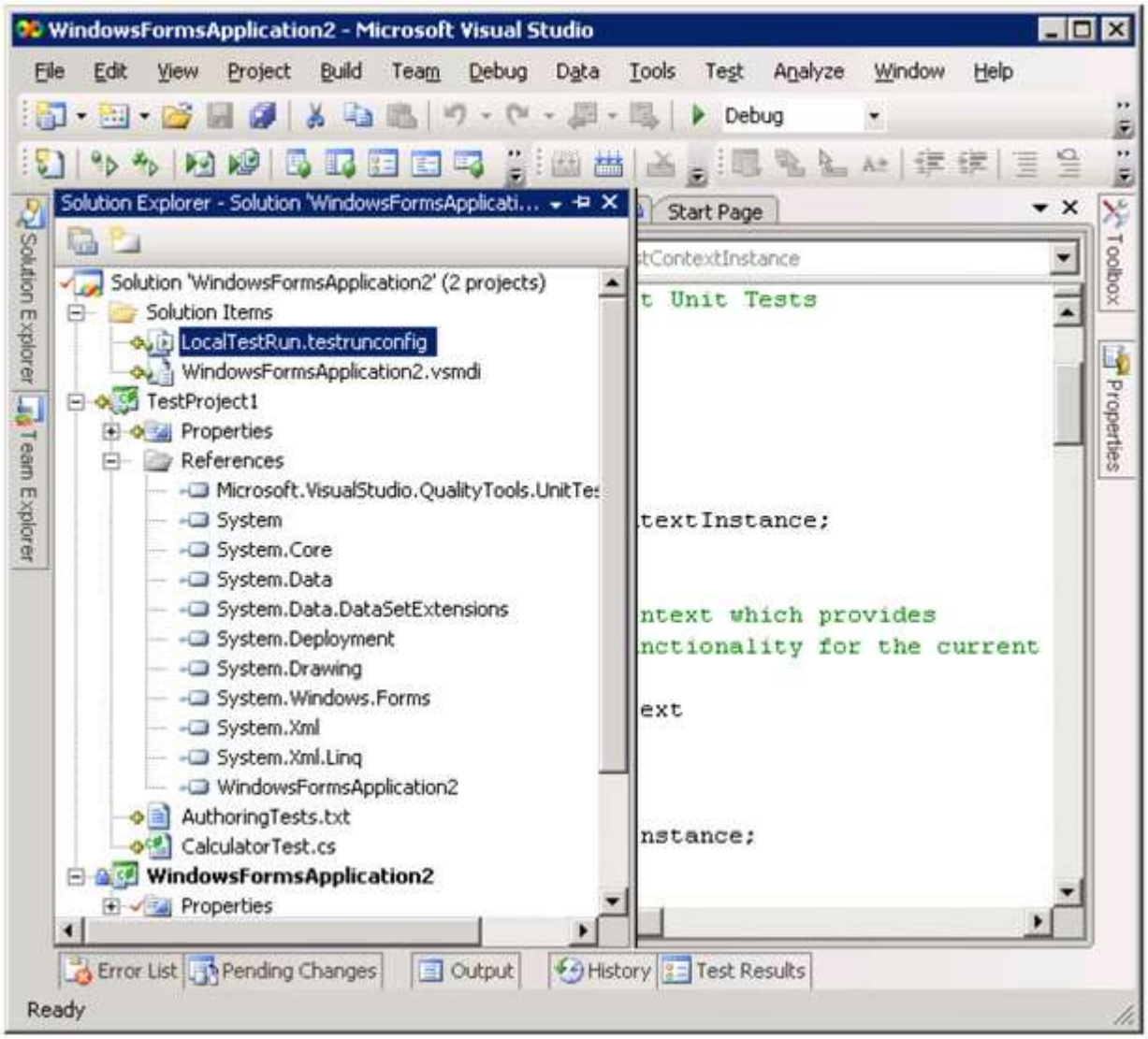
После этого результаты выполнения тестов будут видны в окне результатов:



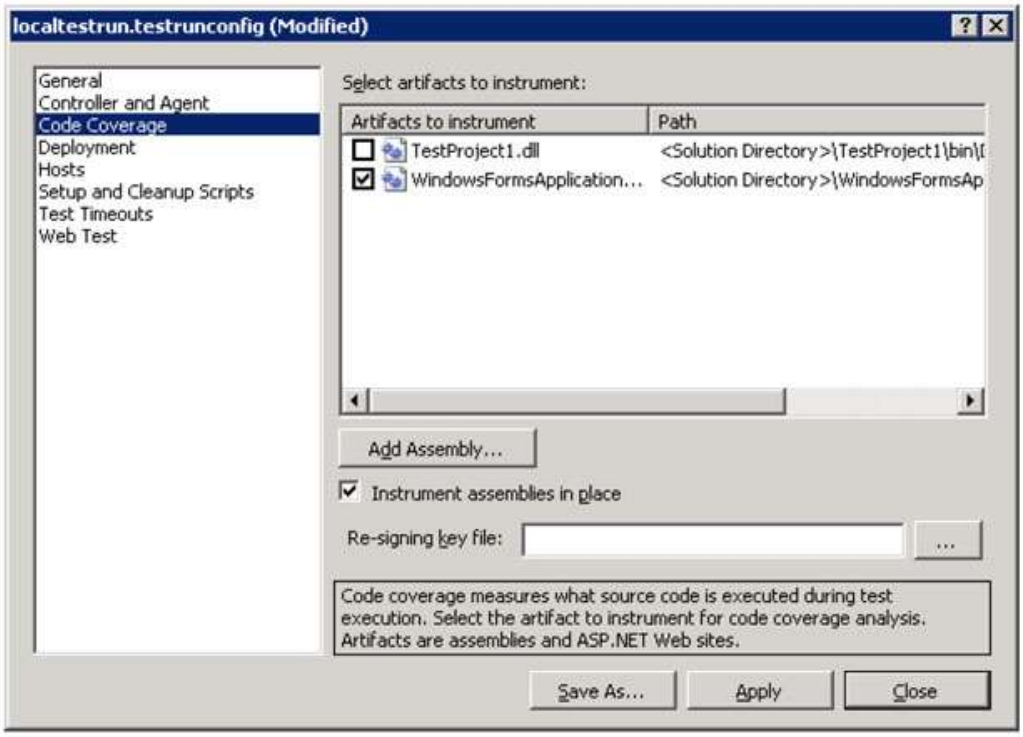
Команды должны добиться того, чтобы все разработанные тесты проходили успешно.

Шаг 4. Изменение конфигурации тестов. Для того чтобы проанализировать качество разработанных тестов, команда должна вычислить тестовое покрытие. Для этого необходимо:

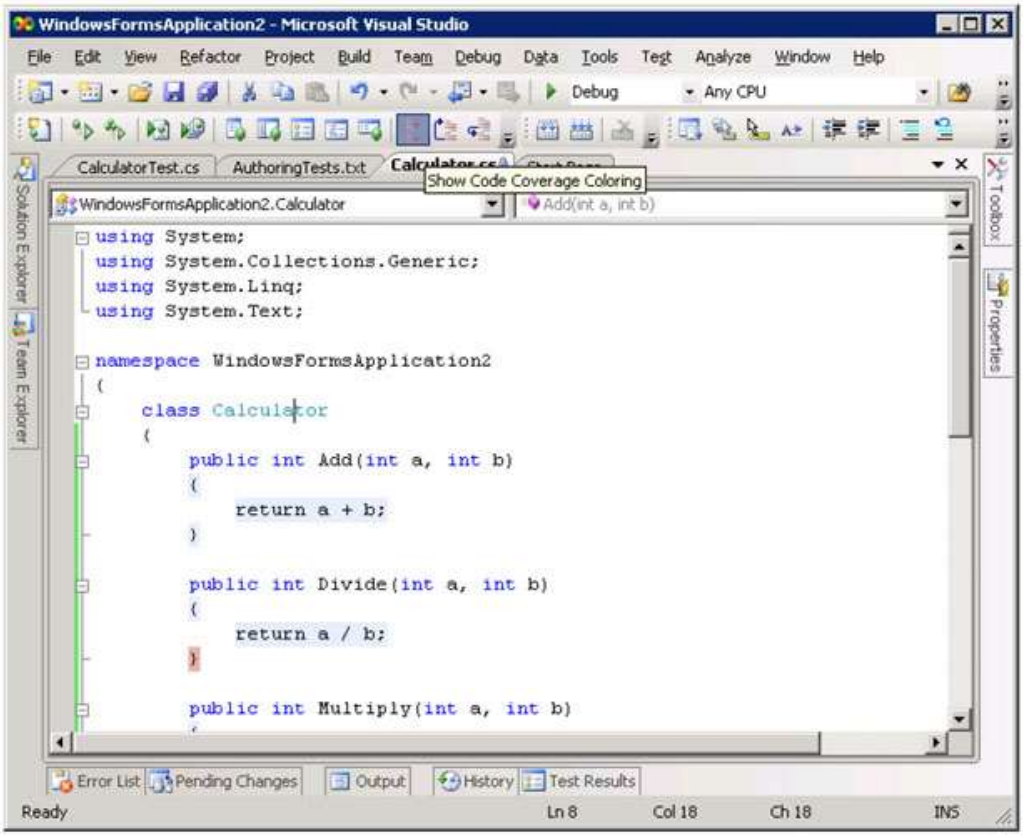
1. Открыть файл конфигурации запуска тестов, автоматически добавленный к решению при создании тестов:



2. В открывшемся диалоге выбрать вкладку Code Coverage и установить то, какие именно проекты нужно анализировать:



3. После сохранения конфигурации запустить тесты и активировать опцию Show Code Coverage Coloring:



ТЕМА 5. СОЗДАНИЕ И КОНФИГУРАЦИЯ АВТОМАТИЧЕСКОЙ СБОРКИ

Цель занятий –создание каждой командой в своем проекте процедуры автоматической сборки. При этом должно быть создано несколько процедур:

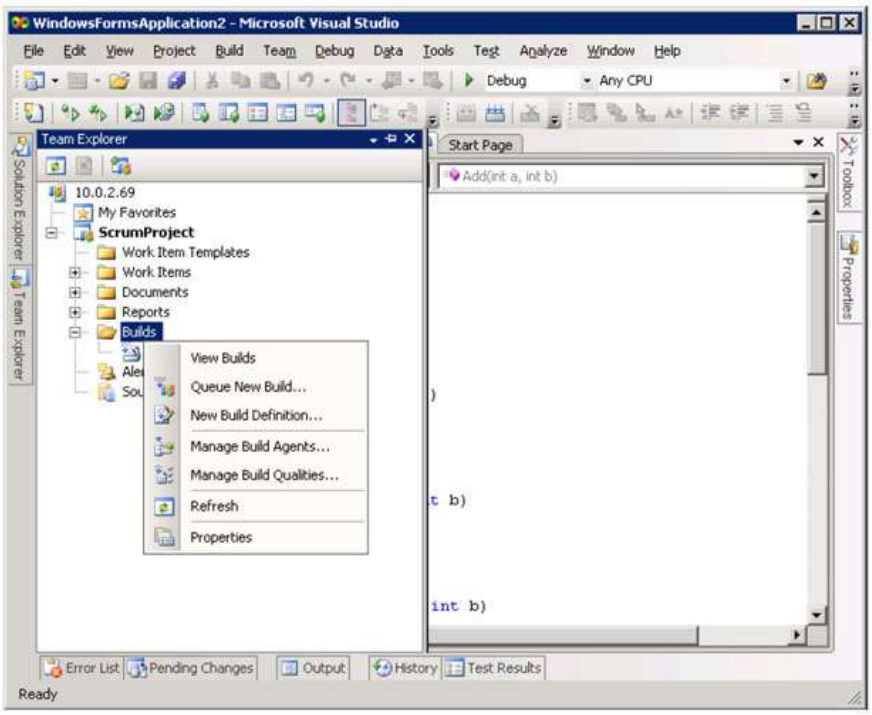
- простая процедура, включающая только сборку;
- полная процедура, включающая тесты и анализ кода.

После создания сборок необходимо настроить параметры непрерывной интеграции:

1. Простая сборка должна запускаться после каждого внесенного изменения, но не чаще чем раз в 5 мин.
2. Полная сборка должна запускаться каждую ночь.

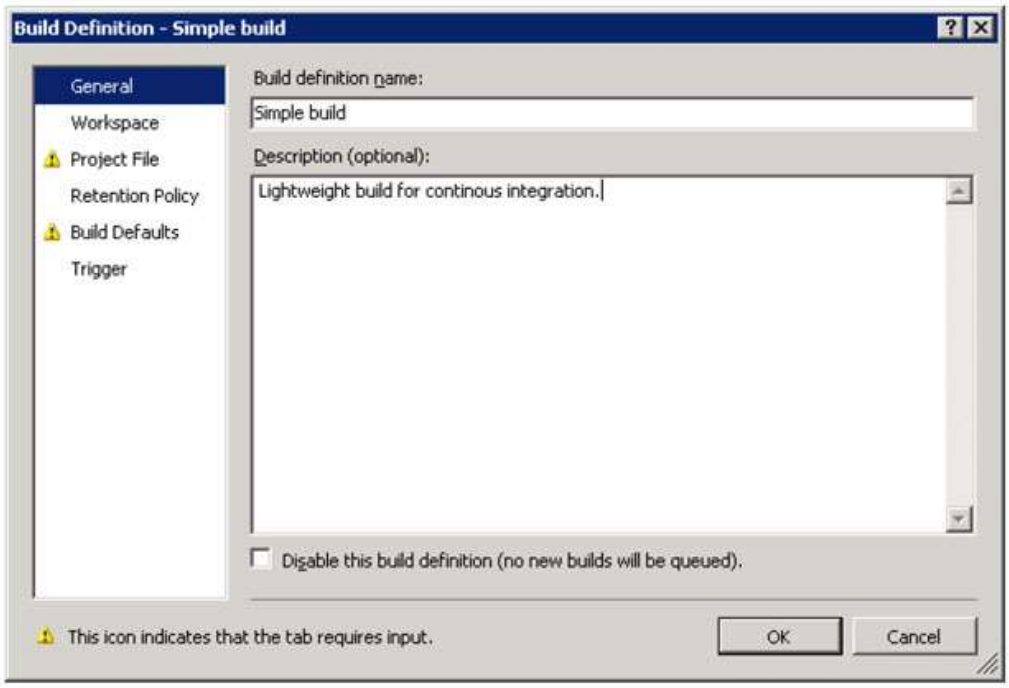
Шаг 1. Создание простой сборки. Все результаты сборки, проведенной TFS, выкладываются в разделяемую папку, на запись в которую есть права у пользователя, с правами которого работает сервер автоматических сборок (обычно – TFSBuild), а также у пользователя, с правами которого работает сам TFS (обычно – TFSService). Как правило, учащиеся не обладают достаточным количеством прав для создания такого рода папок, поэтому они должны быть заранее подготовлены преподавателем.

Для создания простой сборки необходимо обратиться к окну Team Explorer, после чего в разделе Builds выбрать команду Build Definitions:



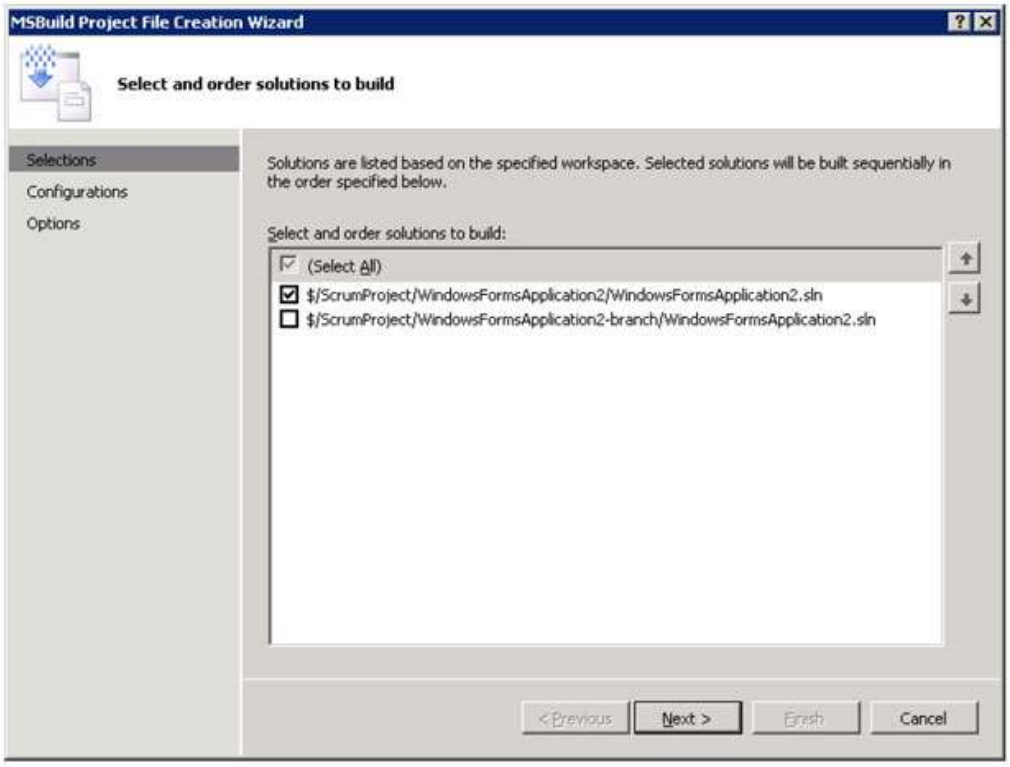
Вызов этой команды приведет к открытию мастера, в котором нужно задать следующие параметры сборки:

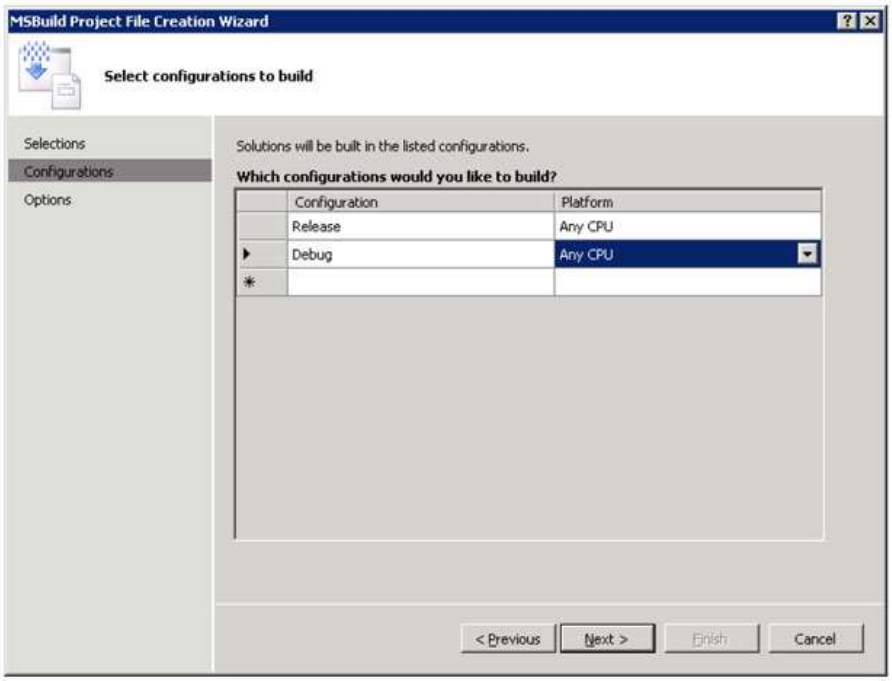
1. Указать имя и описание сборки:



2. На закладке Project File создать новое описание сборки, используя

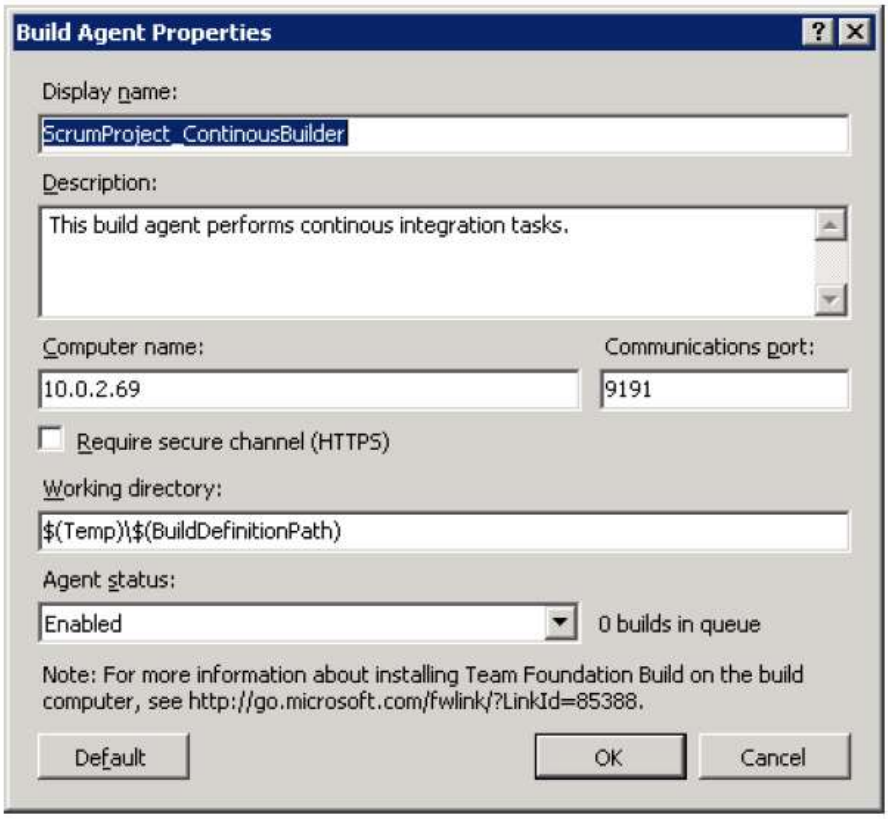
кнопку Create, после чего задать проекты и конфигурации для сборки:



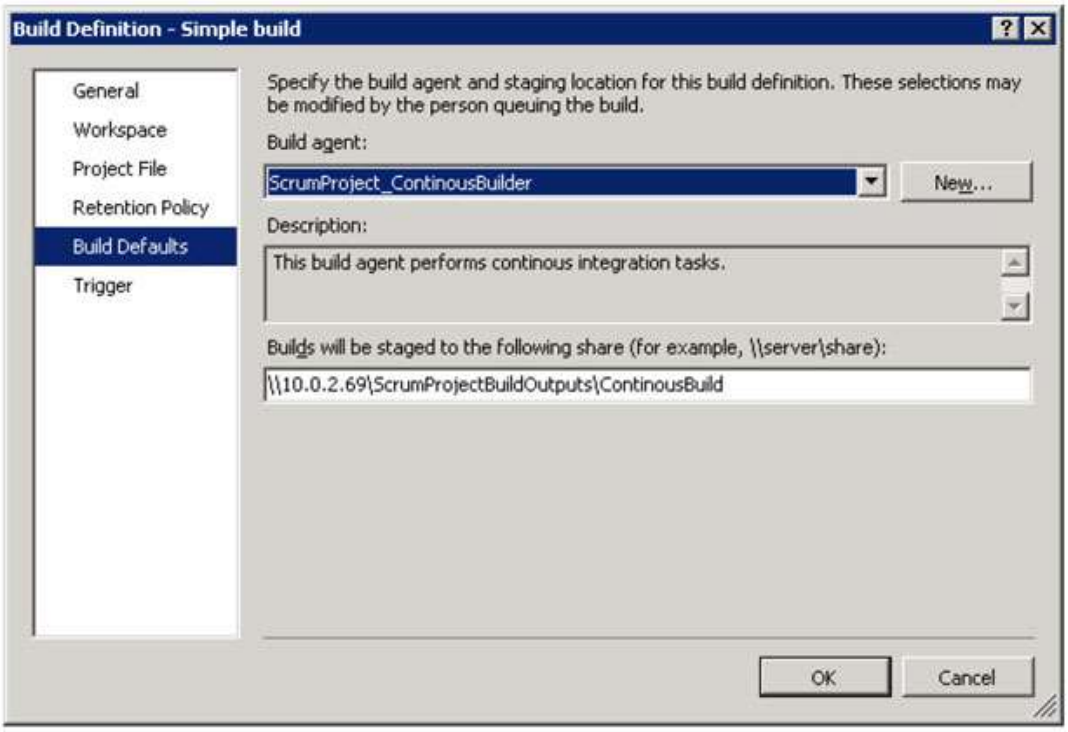


3. На закладке Build Defaults создать определение агента-сборщика, используя кнопку New. Для агента указать имя, описание и IP–адрес сервера

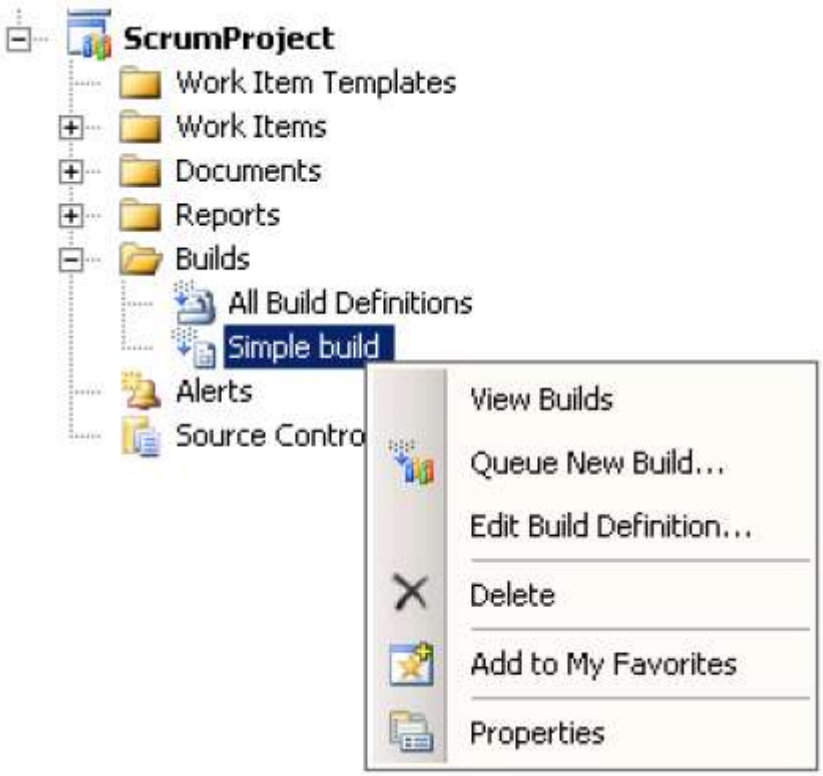
сборок (как правило, совпадает с сервером TFS):



4. На закладке Build Defaults также необходимо задать имя разделяемой папки, в которую будут сложены результаты:



После того как определение сборки было создано, ее необходимо запустить, используя команду Queue new build:

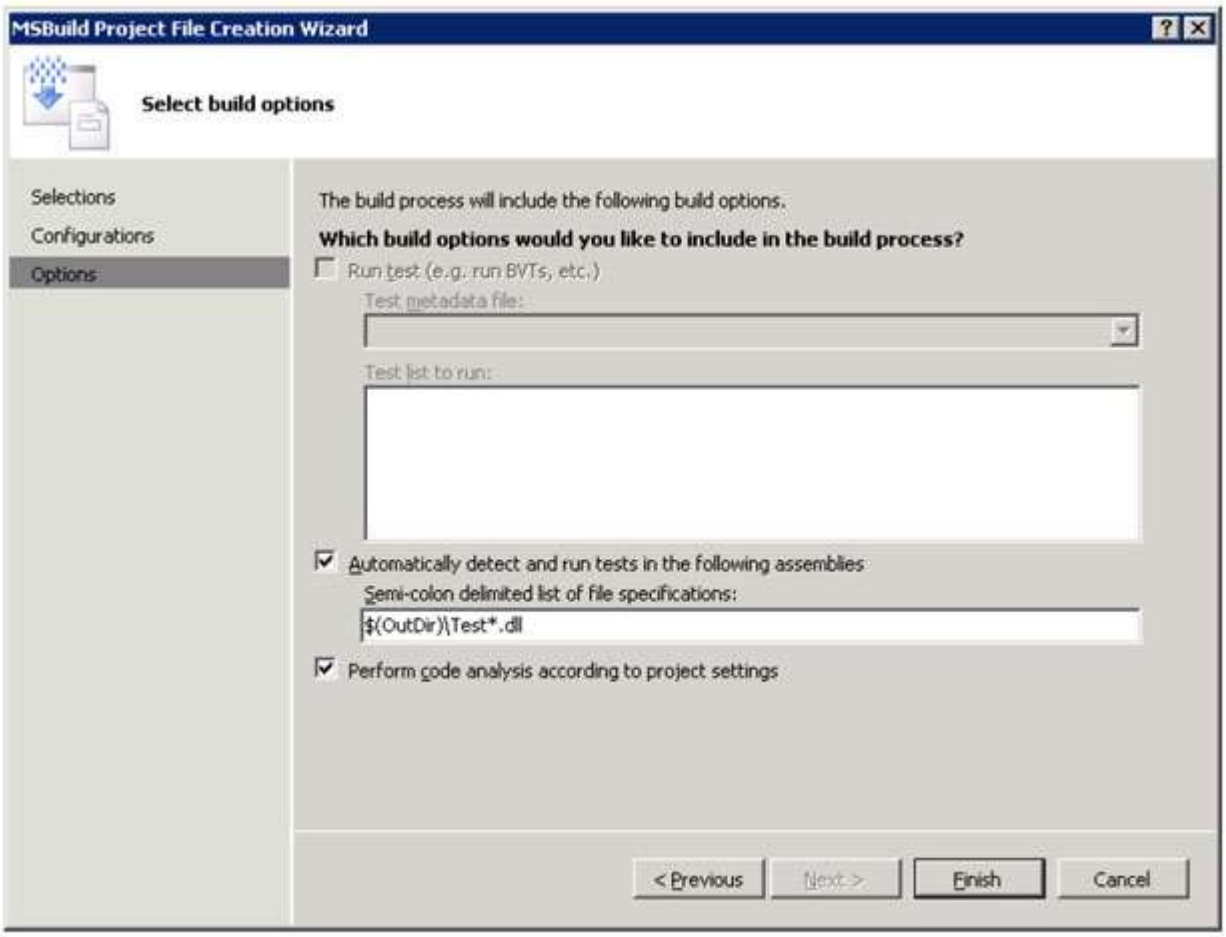


После запуска сборки необходимо дождаться ее завершения и

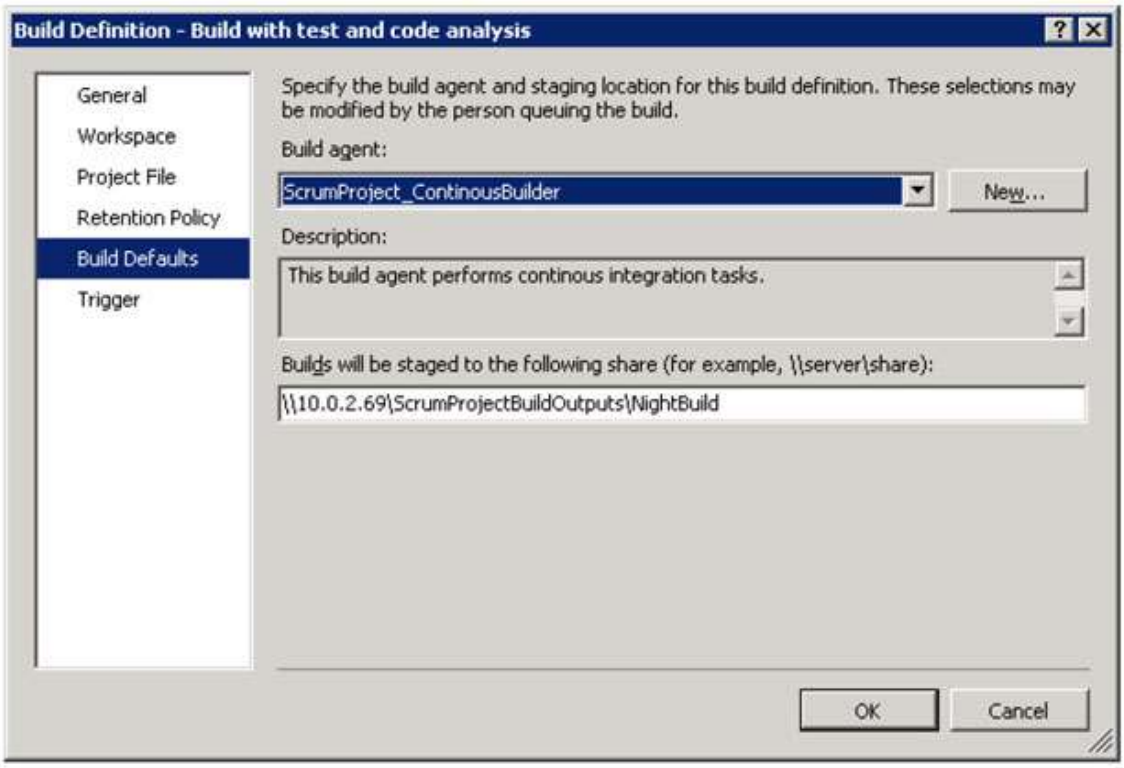
убедиться, что на соответствующей сетевой папке появились результаты сборки.

Шаг 2. Создание сложной сборки. Создание сложной сборки проходит во многом аналогично созданию простой сборки, за исключением нескольких шагов:

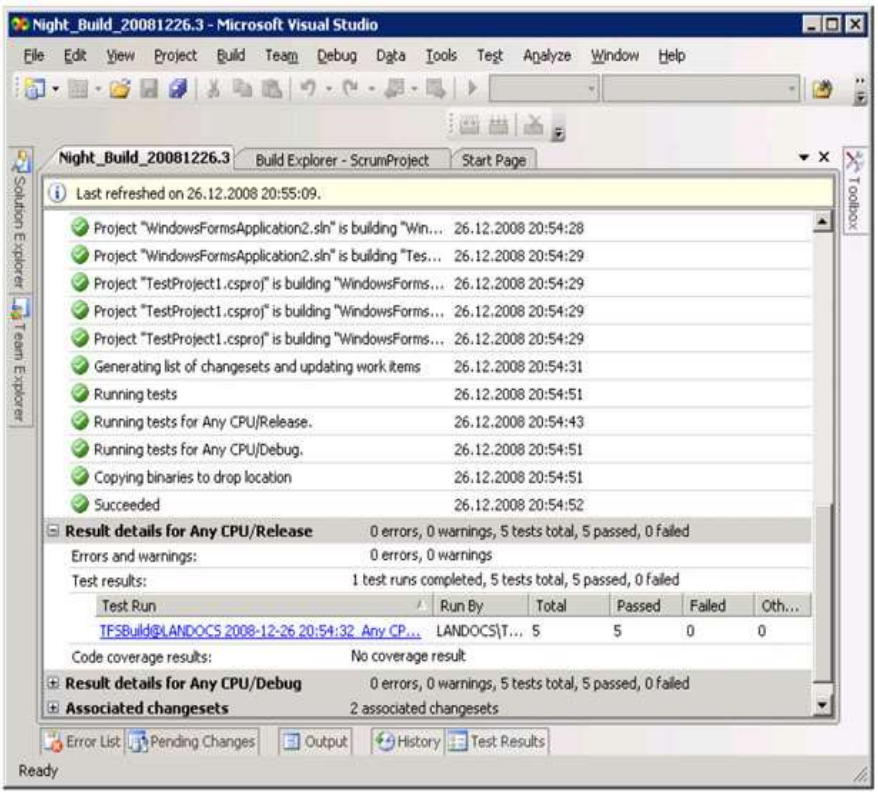
1. На закладке опций при создании проекта сборки необходимо включить автоматический запуск модульных тестов и анализ кода:



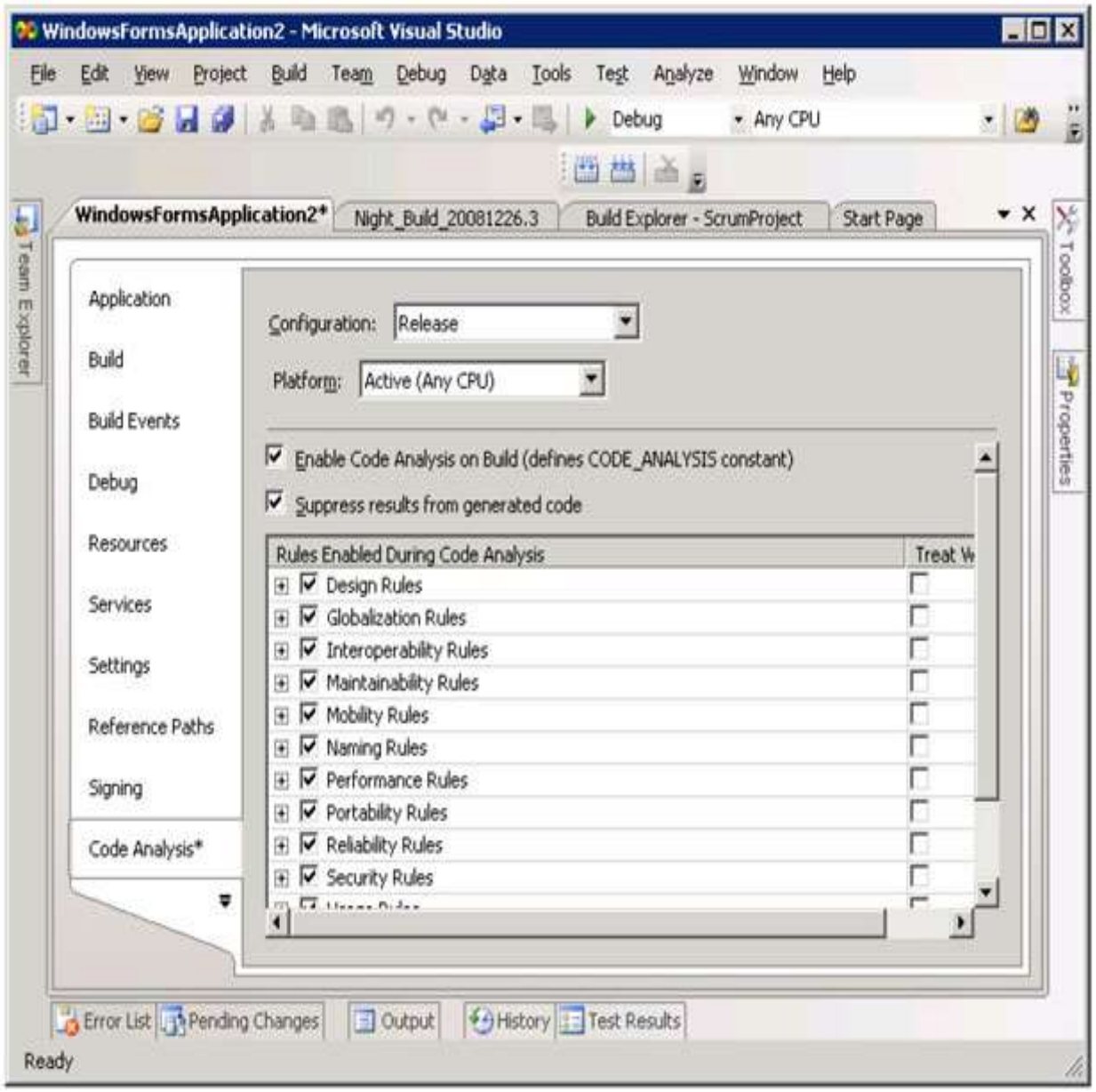
2. На закладке Build Defaults необходимо задать другую папку для сбора результатов:



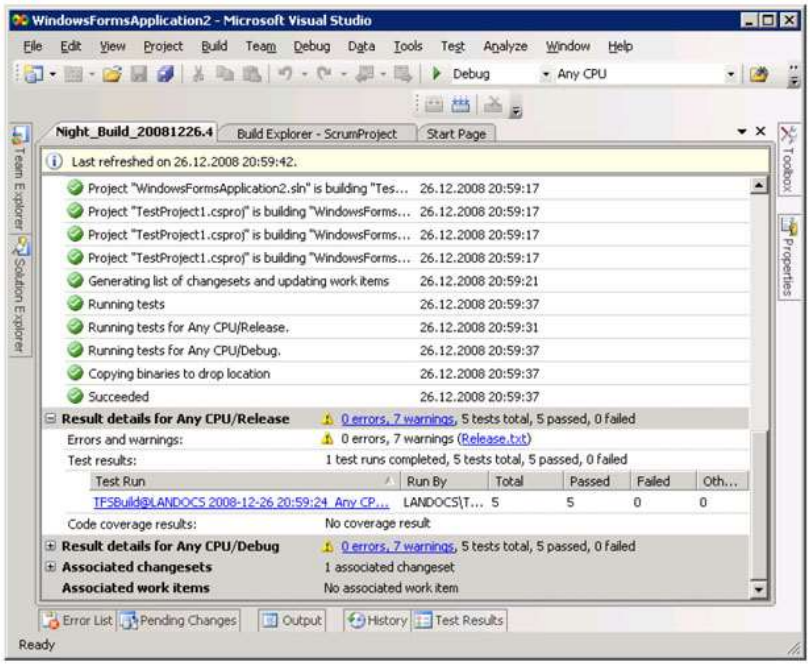
После создания сборки необходимо запустить ее и обратить внимание на то, что результаты сборки включают и результаты тестов:



Теперь нужно добиться выполнения статического анализа кода во время ночной сборки. Для этого необходимо активировать анализ кода в настройках соответствующих проектов:

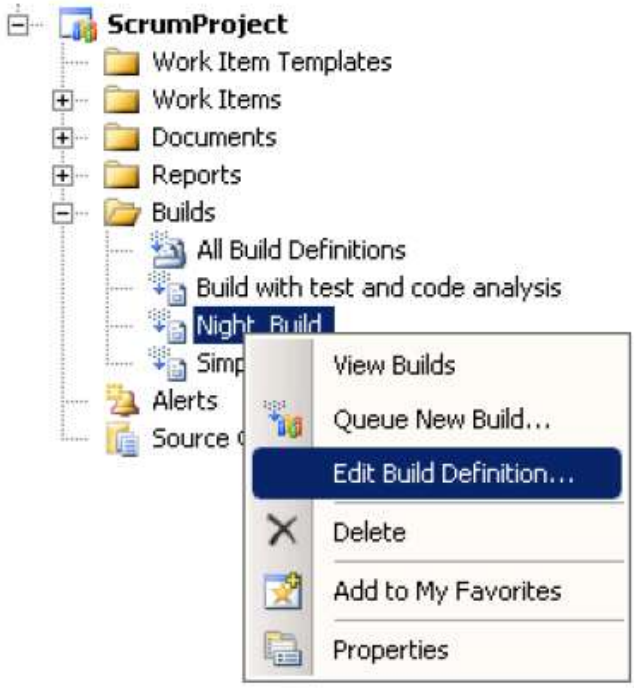


Следующая же собранная сборка будет содержать большое количество предупреждений от анализатора:

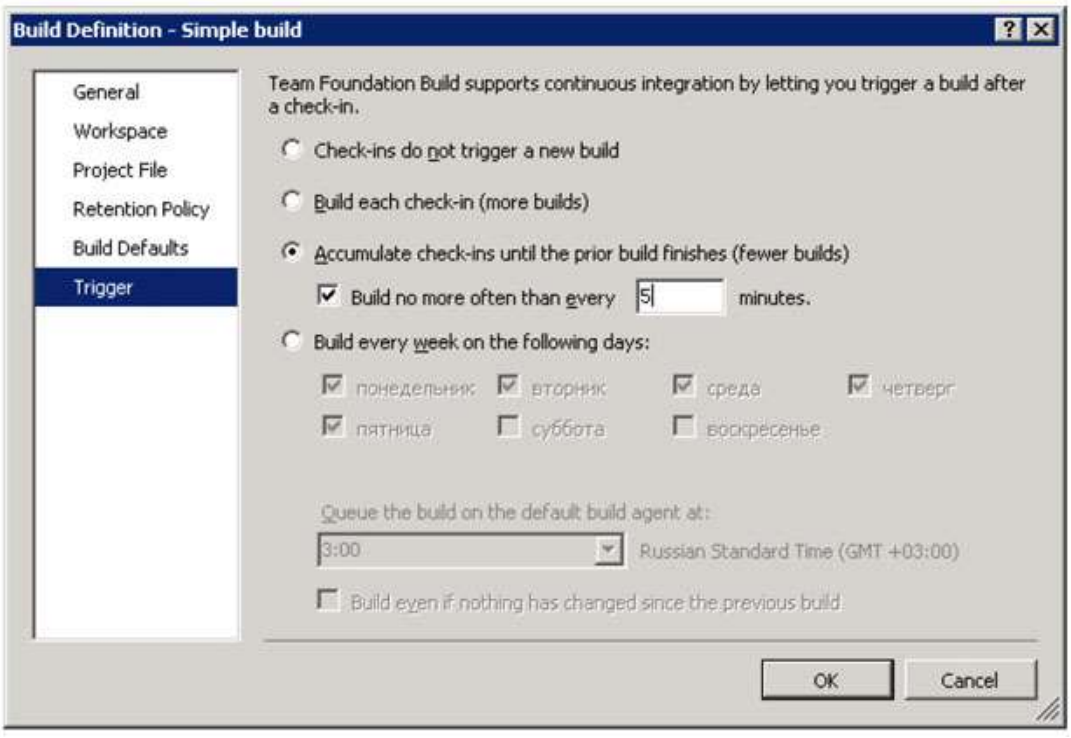


Шаг 3. Настройка непрерывной интеграции. На данном шаге учащимся необходимо исправить описания сборок таким образом, чтобы они выполнялись автоматически при определенных условиях. Простой вариант сборки должен запускаться автоматически после каждого внесения изменений, но не чаще, чем раз в пять минут. Для того чтобы добиться этого, необходимо:

1. Вызвать команду Edit build definition:

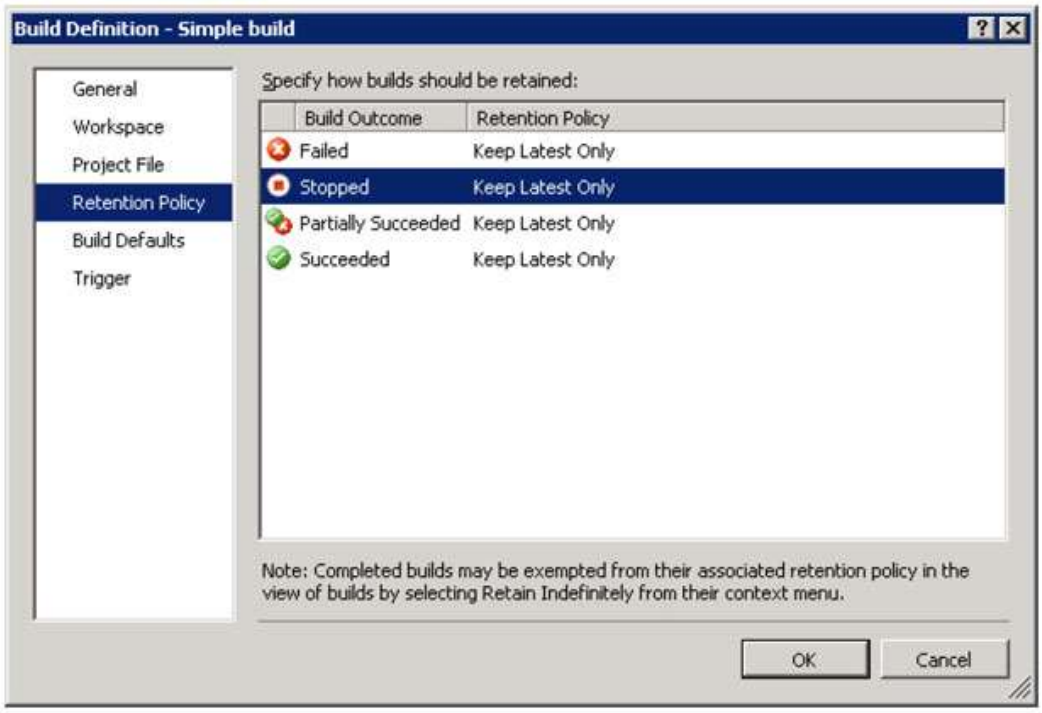


2. Задать настройки автоматического запуска на закладке Trigger:



3. Настроить политику очистки сборок на закладке Retention Policy. Это необходимо для того, чтобы избежать быстрого исчезновения места на машине-сборщике и для удаления из базы TFS информации о

второстепенных сборках:

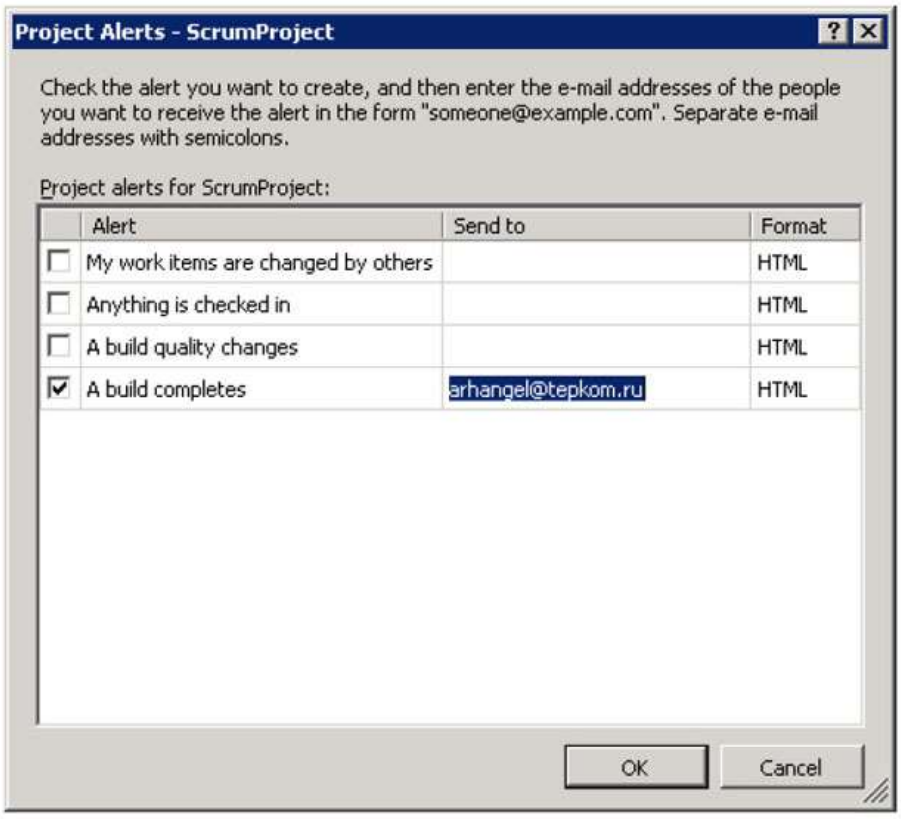


Проведение сборки после внесения изменений наиболее эффективно в том случае, если участники проекта получают нотификации о том, что сборка была проведена. Для того чтобы этого добиться, необходимо:

1. В контекстном меню проекта выбрать команду Project Alerts:

2. В списке событий, о которых нужно слать нотификации, выбрать «a build completes» и задать список адресов электронной почты, на которые

нужно отправить сообщение:



После настройки простой сборки для запуска при внесении изменения необходимо проверить работу системы: внести некоторое изменение и дождаться сообщения о сборке.

Аналогичным образом можно настроить и автоматический запуск сложной сборки каждый день в определенное время.

ТЕМА 6. НАСТРОЙКА ШАБЛОНА ПРОЦЕССА

Цель занятия – провести ретроспективу своей деятельности и сформировать список возможных изменений в процессе и работе с TFS. Внесение замечаний в TFS как соответствующие элементы работы.

В рамках ретроспективы команда должна предложить некоторые изменения к элементам работы, вовлеченным в процесс, а затем и реализовать эти изменения.

Шаг 1. Ретроспектива. Команда в течение 20–30 мин. обсуждает то,

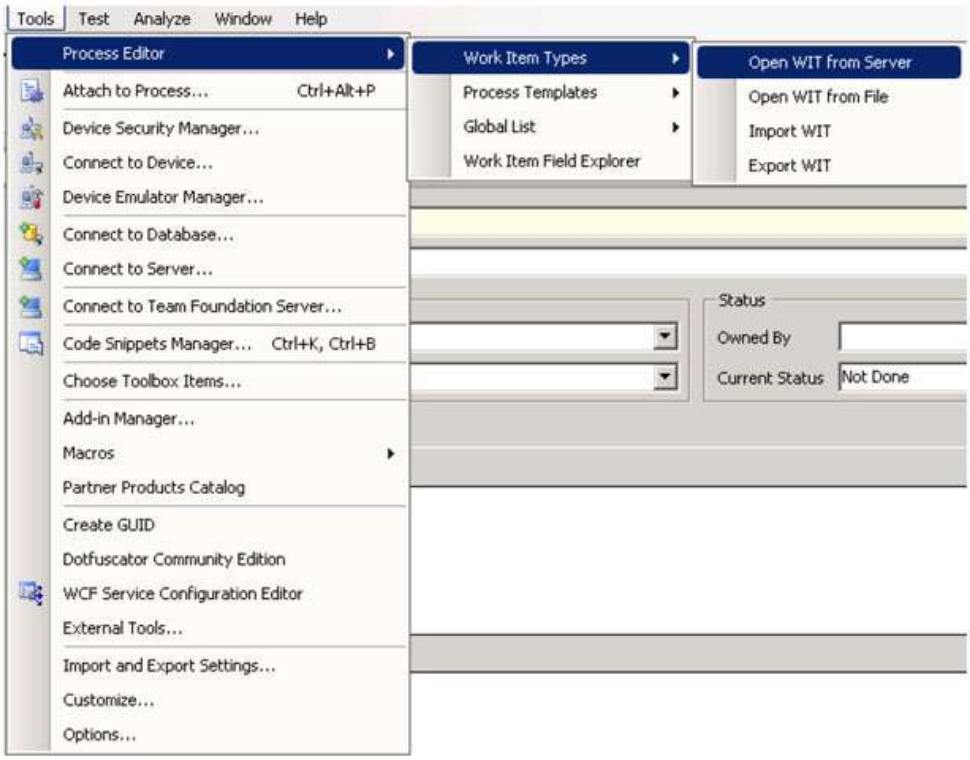
как прошел данный спринт, выделяя позитивные и негативные моменты, а также предложения по изменениям.

Все комментарии должны быть внесены в соответствующий элемент работы типа Sprint retrospective, а для каждого предложения по улучшению заведены элементы работы типа Sprint backlog item, а для каждого идентифицированного негативного момента, требующего устранения, – элемент работы типа Impediment.

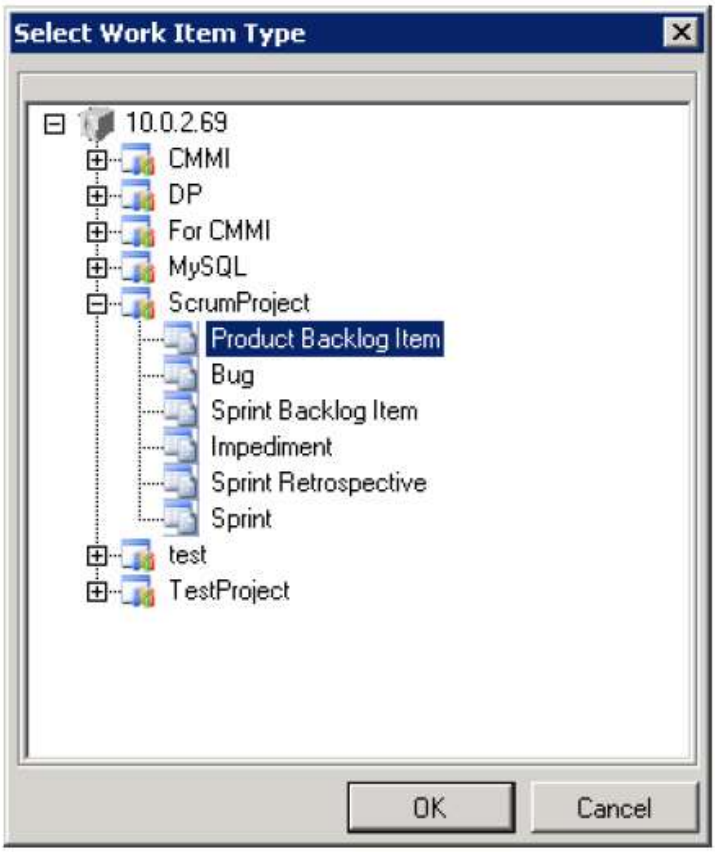
Результаты ретроспективы необходимо обсудить с хозяином продукта.

Шаг 2. Изменение элемента работы. На этапе ретроспективы команда должна выявить некоторые изменения в формате и жизненном цикле элементов работы, которые помогут повысить эффективность команды. На следующем шаге им нужно воплотить эти изменения в жизнь. Для этого необходимо:

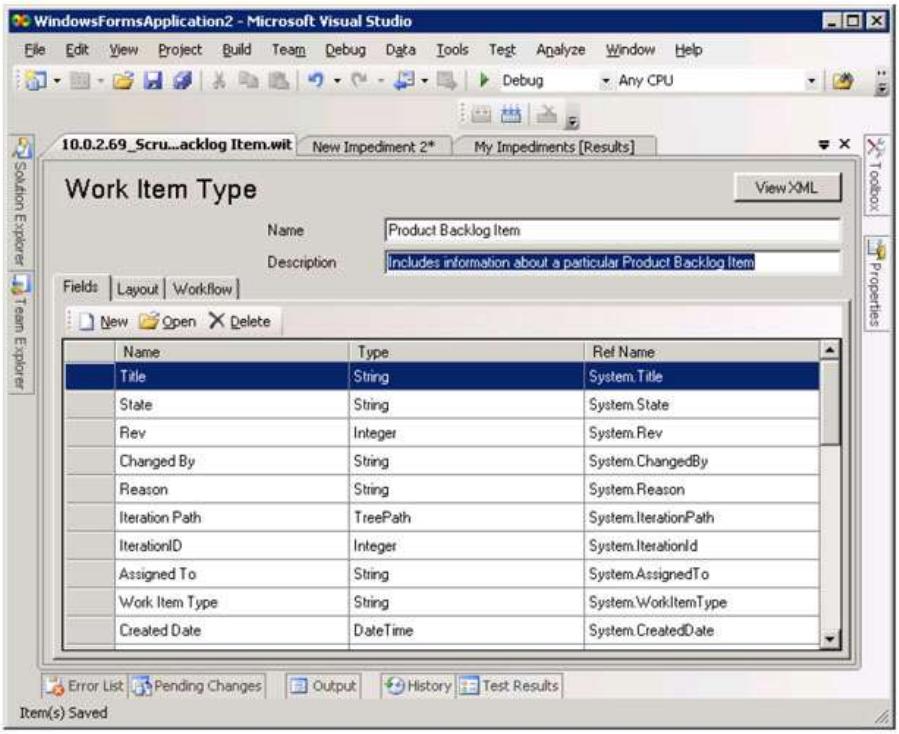
1. Открыть тип элемента работы на редактирование с помощью команды меню Tools:



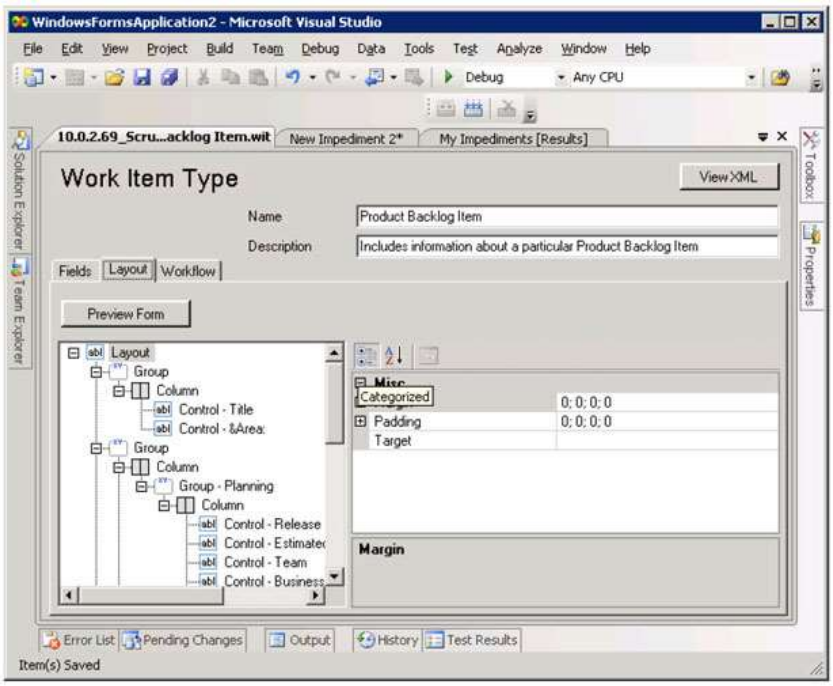
2. Выбрать нужный элемент работы в открывшемся диалоге:



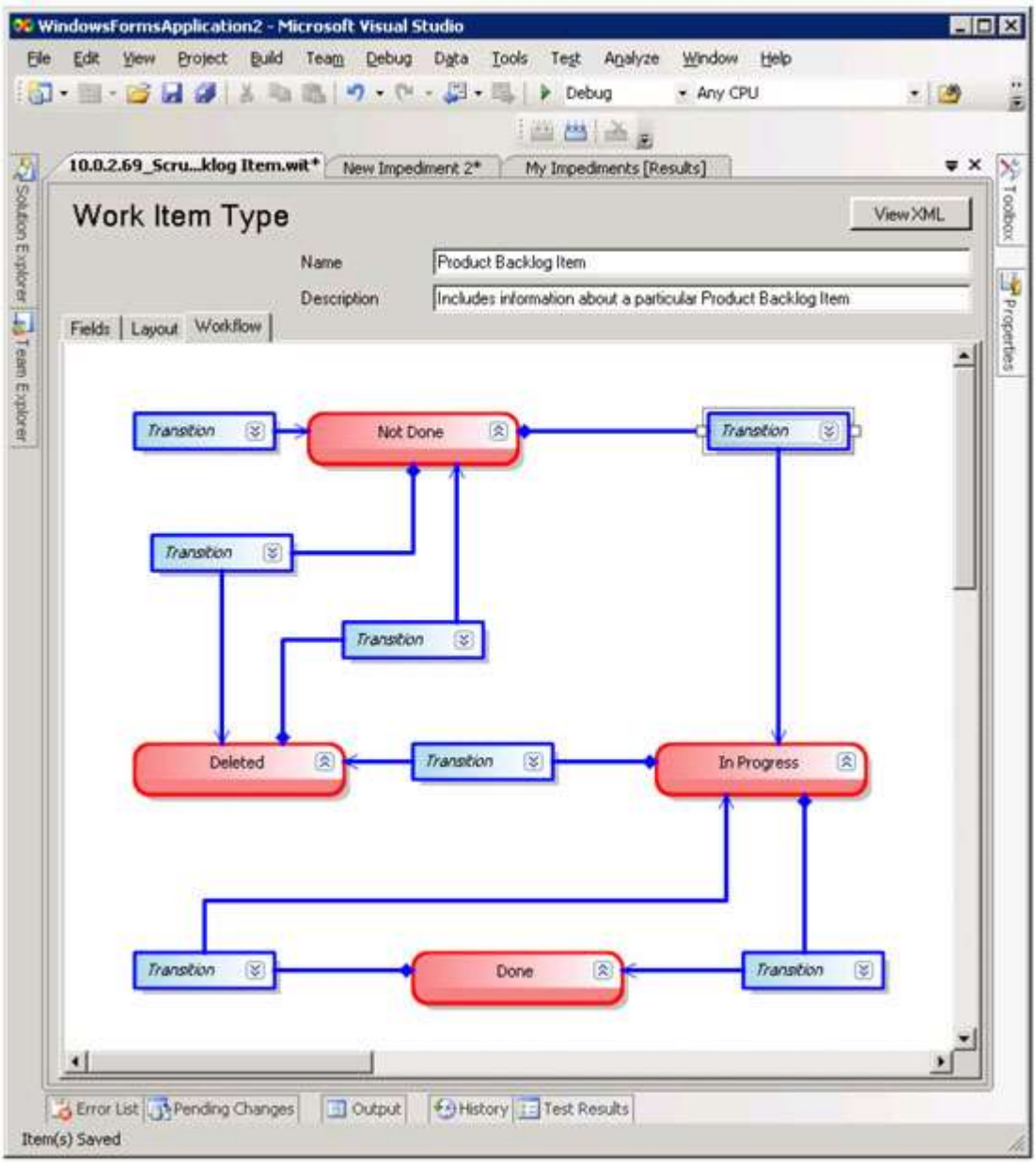
3. На закладке Fields добавить, удалить или изменить необходимые поля:



4. На закладке Layout изменить соответствующим образом визуальное представление элемента работы:



5. На закладке Workflow внести необходимые изменения в жизненный цикл элемента работы:



После внесения всех изменений их нужно сохранить, а затем убедиться, что они применились к существующим и к вновь создаваемым элементам работы.

ЗАКЛЮЧЕНИЕ

Современная программная инженерия — молодая и быстро развивающаяся область знаний и практик. Она ориентирована на комплексное решение задач, связанных с разработкой особой разновидности сложных систем — программных систем.

Программные системы — самые необычные и удивительные создания рук человеческих. Они не имеют физических тел, их нельзя потрогать, ощутить одним из человеческих чувств. Они не подвергаются физическому износу, их нельзя изготовить в обычном инженерном смысле, как автомобиль на заводе. И вместе с тем разработка программных систем является самой сложной из задач, которые приходилось когда-либо решать человеку-инженеру. В пределе — это задача создания рукотворной системы, сопоставимой по сложности самому творцу.

Многие стереотипы и приемы, разработанные в физическом мире, оказались неприменимы к инженерии программных систем. Приходилось многое изобретать и придумывать. Все это теперь история. Программные инженеры начинали с полного неприятия инструментария других инженерных дисциплин, уверовав в свою кастовость «жрецов в белых халатах», и совершили эволюционный круг, вернувшись в лоно общечеловеческой инженерии.

Впрочем, были времена, когда и другие «кланы» людей относились к «программам» с большим подозрением: им мало платили, унижая материально, не находили для них ласковых слов (а употребляли большей частью ругательные). Где эти люди? И где их прежнее отношение?

Современное общество впадает во все большую зависимость от программных технологий. Программного инженера стали любить, охотно приглашать в гости, хорошо кормить, обувать и одевать. Словом, стали лелеять и холить (правда, время от времени продолжают сжигать на костре и предавать анафеме).

Современная программная инженерия почти достигла уровня зрелости — об этом свидетельствуют современные тенденции; она разворачивается от сердитого отношения к своим разработчикам к дружелюбному, снисходительному пониманию человеческих слабостей.

Базис современной программной инженерии рассмотрен в данном

учебнике. Конечно, многое осталось за кадром. Реорганизация (рефакторинг), особенности конструирования web-приложений, работа с базами данных — вот неполный перечень тем, обсудить которые не позволили ресурсные ограничения. Хотелось бы обратить внимание на новейшие постобъектные методологии — аспектно-ориентированное и многомерное проектирование и программирование. Они представляют собой новую высоту в стремительном полете в компьютерный космос.

Библиографический список

1. Орлов, С. А. Технологии разработки программного обеспечения: Разработка сложных программных систем / С. А. Орлов. – 3-е изд. – СПб.: Питер, 2004.
2. Кузнецов, Д. В. Введение в программную инженерию [Электронный ресурс]: Курс лекций / Д. В. Кознов, Д. Ю. Бугайченко. – Режим доступа: http://www.intuit.ru/department/se/inprogeng/
3. Кузнецов, Д. В. Опыт сочетания теории и практики в обучении программной инженерии / Д. В. Кознов, Я. А. Кириленко. III Междунар. науч.-практ. конф. Современные информационные технологии и ИТ-образование. – Режим доступа: http://2008.itedu.ru/pages/Conference-works.
4. Koznov, D. V., M. Y. Pliskin. Computer-Supported Collaborative Learning with Mind-Maps. T. Margaria and B. Steffen (Eds.): ISoLA 2008, CCIS 17, pp. 478–489, 2008. Springer-Verlag, Berlin Heidelberg, 2008.
5. Кознов, Д. В. Методика обучения программной инженерии на основе карт памяти. Системное программирование / Д. В. Кознов // Вып. 3. под ред. А. Н. Терехова и Д. Ю. Булычева. – СПб.: Изд.-во СПбГУ, 2008. -с. 121–140.
6. Антамошкин, О. А.. Разработка и стандартизация программных средств и информационных технологий : учеб. пособие / О. А. Антамошкин; Сиб. гос. аэрокосмич. ун-т. – Красноярск, 2008.

Приложение. Игра «Балда»

Игровое поле «Балды»

	р	д		
б	а	л	д	а
	к			

1 игрок
лак (3)
бард (4)

2 игрок
бар (3)

Балда — довольно простая и распространенная настольная игра, в которой необходимо составлять слова с помощью букв, расположенных на игровом поле. Принимать участие могут два и более игроков. Существует большое количество компьютерных вариантов игры. Правила игры были напечатаны Э. Иодковским в журнале «Наука и жизнь».

Описание

Как правило, игровое поле представляет таблицу из 25 клеток (5x5), в центральный горизонтальный ряд которого ставится произвольное слово из пяти букв, каждая буква которого находится в отдельной ячейке. Размеры поля, расположение слова, длину слова можно варьировать, однако необходимо, чтобы количество пустых клеток в начале игры было четным,

чтобы у обоих игроков получилось одинаковое количество слов. Для игры на бумаге необходимо начертить на листке в клеточку квадрат (обычно 5х5, но возможно 7х7 и т.д.). Далее один из игроков придумывает слово с количеством букв, равным стороне квадрата, и вписывает его в середину квадрата.

Во время своего хода игрок должен подставить на игровое поле букву так, чтобы она располагалась в клетке, смежной с уже заполненными клетками по горизонтали или по вертикали. Иными словами, слева, справа, сверху или снизу относительно уже заполненных клеток. (Существует также «королевский» вариант, который предусматривает установку букв по диагонали). После этого необходимо составить слово с использованием установленной буквы. При этом действуют следующие правила:

- Слово должно составляться переходом по смежным клеткам, расположенным под прямыми углами относительно друг друга (в «королевском» варианте — в любых смежных направлениях).
- Слово должно существовать в словарях.
- Слово должно являться нарицательным именем существительным в начальной форме (единственном числе и именительном падеже или множественном числе и именительном падеже в случае слова, не употребляющегося в единственном числе).
- В слове должна быть использована поставленная на поле буква. Иногда правила допускают прохождение по одной и той же клетке дважды, т.е., например, из букв «к», «о», «л» можно составить слово «колокол». Например, к слову «балда» первый игрок приписывает букву «к». Получилось слово «лак».
- Чем длиннее слово вы придумаете, тем больше очков получаете. Одна буква — одно очко.
- Игра заканчивается тогда, когда будут заполнены все клетки.
- В каждой клетке должно быть по 1 букве, игроки ходят по очереди.
- Слова в одной игре повторяться не могут.
- По предварительной договоренности буквы «е» и «ё», а также иногда «и» и «й» считаются за одну и ту же букву.
- Запрещается написание слова в случае, если оно уже читается на поле, даже если оно не было составлено ни одним из играющих.
- В королевском варианте запрещаются любые самопересечения слова.

Устный вариант

- Играть могут два или более человек.
- Разрешается использовать только нарицательные существительные в единственном числе и именительном падеже. Жаргонные и составные слова запрещены.
- На всю игру устанавливается определенная очередность ходов и количество штрафных очков, до которых ведется игра. Начинающий называет любую букву.
- Следующий игрок либо делает ход, либо отказывается от хода. Ход заключается в добавлении своей буквы до или после данной ему буквы (буквосочетания), причем нельзя создавать сочетания, не встречающиеся в русском языке, а также сочетания, образующие законченное слово.
- Если игрок отказался от хода, а предыдущий игрок не нарушил правил, то отказавшийся игрок получает штрафное очко. Если же после отказа от хода выясняется, что правила были нарушены, то штрафное очко получает нарушитель.
- После присуждения штрафного очка игра продолжается с нового слова, следующий по очереди игрок называет первую букву.

Возможные модификации:

- Разрешается подставлять очередную букву не только по краям, но и внутрь текущего буквосочетания.
- Разрешается переставлять уже названные буквы, т.е. использовать анаграммы. В этом варианте сказанные буквы рассматриваются не как фрагмент слова, а как множество.
- «Антибалда»: тот, кто первым завершает слово, зарабатывает выигрышное очко. В этом варианте игры можно набрать несколько очков, пока не будут исчерпаны возможности продолжения.