МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ВОЛОГОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

А.В. Машкин

ТЕХНОЛОГИЯ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Утверждено редакционно-издательским советом ВоГУ
в качестве учебного пособия

Вологда
2014

ББК 32.973.2-018
УДК 681.3.06
М 38

Рецензенты:
О.А. Шахов, кандидат технических наук,
начальник инженерно-экономического факультета
Вологодского института права и экономики,

В.Г. Родимов, директор ООО «Мультимедиум»

Машкин, А.В.
М 38 **Технология разработки программного обеспечения**: учебное пособие / А.В. Машкин. – Вологда: ВоГУ, 2014. – 75 с.

ISBN 978-5-87851-526-9

В учебном пособии рассматриваются основы процесса разработки программного обеспечения. Также произведен обзор наиболее популярных технологий разработки программного обеспечения, парадигм и методологий, лежащих в их основе. Особое внимание уделяется технологии командной разработки программного обеспечения с использованием инструментальных средств Microsoft (TFS 21012 и VS2012). Пособие подготовлено на основе программы курса «Технология разработки программного обеспечения» для студентов магистратуры направления 230100.68 «Информатика и вычислительная техника».

ББК 32.973.2-018
УДК 681.3.06

ISBN 978-5-87851-526-9

ВВЕДЕНИЕ

Разработка программного обеспечения (ПО) представляет из себя вид деятельности, основной целью которого является создание работоспособного ПО, обеспечения его соответствия заданным критериям качества и надежности. При разработке ПО достаточно активно применяются технологии, методологии и практики из смежных областей знаний, таких как информатика, математика, управления проектами и др.

Точно так же, как и в других инженерных дисциплинах при разработке ПО приходится иметь дело с проблемами надежности, стоимости и качества. Исходные тексты некоторых программных приложений содержат миллионы строк, при этом ожидается, что такое ПО должно корректно функционировать в изменяющихся условиях.

В разработке ПО можно выделить следующие разделы [1]:

1) Выработка требований к разрабатываемому ПО (извлечение, анализ, спецификация и ратификация требований).

2) Проектирование и описание ПО с использованием средств автоматизированной разработки ПО (CASE) и стандартов форматов описаний, таких как, например, унифицированный язык моделирования (UML). При проектировании ПО используются различные подходы, например, проблемно-ориентированное проектирование и т.д.

3) Инженерия ПО, т.е. его создание с помощью языков программ-мирования.

4) Тестирование ПО (поиск и исправление ошибок в программе).

5) Обслуживание ПО (Программные системы часто имеют проблемы совместимости и переносимости, а также нуждаются в последующих модификациях в течение долгого времени после того, как закончена их первая версия).

6) Управление конфигурацией ПО (так как системы программного обеспечения очень сложны и модифицируются в процессе эксплуатации, их конфигурации должны управляться стандартизированным и структури-рованным методом).

7) Управление разработкой ПО.

8) Процесс разработки ПО – это построение ПО с использованием определенной парадигмы проектирования, среди которых в настоящее время считаются agile (гибкая модель разработки) и waterfall (водопадная или каскадная модель).

9) Инструменты разработки программного обеспечения.

10) Оценка качества ПО (выработка критериев, по которым осущест-вляется оценка качества и надежности ПО, а также разработка методик для определения количественных показателей этих критериев).

В данном учебном пособии в основном рассматриваются вопросы, посвященные организации процесса разработки ПО.

1. ОСНОВЫ ПРОЦЕССА РАЗРАБОТКИ ПО

При попытке перейти непосредственно к этапу написания кода при разработке сложных программных комплексов без предварительных этапов выработки требований к разрабатываемой системе, анализа и проектирования, даже в том случае, если удается создать работающее ПО, то будут иметь место следующие негативные факторы:

1) Много лишнего объема проделанной работы. Из-за отсутствия четко поставленных задач придется несколько раз переписывать один и тот же код, уже в ходе этого процесса, формулируя точную постановку задачи.

2) Следствием первого пункта будет являться срыв сроков выполнения разработки ПО.

3) Разработанное таким образом ПО будет отличаться низким качеством (низкая надежность программы, большая вероятность выявления дополнительных ошибок на этапе эксплуатации программы).

4) Весьма значительные трудности при сопровождение такого ПО.

Поэтому разработка больших программных комплексов осуществляется в несколько этапов. В качестве основных этапов разработки надежного, безопасного и быстродействующего ПО можно выделить:

1) Постановку задачи.
2) Выработку требований к разрабатываемой системе.
3) Анализ и разработку архитектуры системы.
4) Уточнение архитектуры.
5) Кодирование и отладку.
6) Тестирование.
7) Внедрение.
8) Сопровождение.

На первом этапе четко формулируются задачи, которые должен решать разрабатываемый программный комплекс. На втором этапе происходит выработка требований к разрабатываемому ПО (ограничения, накладываемые на ПО, условия функционирования ПО – на какой программно-аппаратной платформе оно будет работать, однопользовательская или многопользовательская система, максимальное количество пользователей в системе и т.д.). Первые два пункта, как правило, формулируются заказчиком и траты времени со стороны разработчиков ПО при их работе на эти пункты не происходит.

На рис. 1.1 приблизительно показана доля каждого из этапов в общем объеме времени, затрачиваемых разработчиками на работу над реализацией ПО. Этап внедрения, точно так же, как и два первых этапа, как правило, осуществляются заказчиком.

На этапе предварительного анализа происходит анализ требований к разрабатываемой системе. Поскольку изначально задача ставится в терминах предметной области, то необходимо перевести ее в термины близкие к программированию. Поскольку программист, как правило, редко разбирается

в предметной области, а заказчик – в программировании, то постановка задачи может стать весьма непростым итерационным процессом.

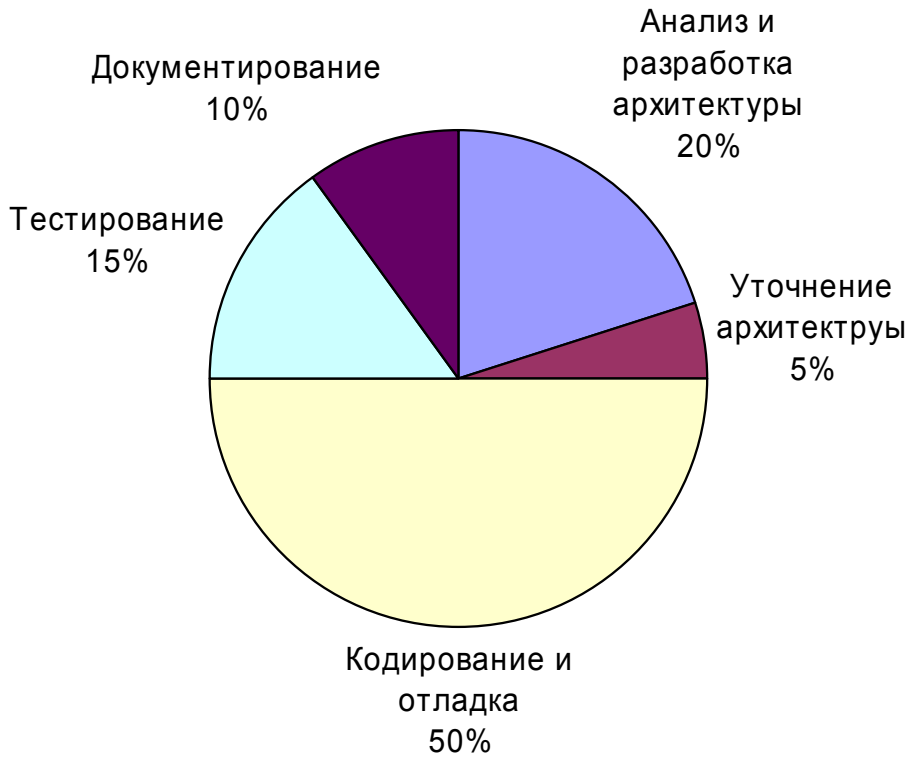

Рис. 1.1. Процент каждого из этапов в общем объеме времени разработки ПО

С другой стороны у заказчика может и не быть свободного времени для общения с командой разработчиков, в этом случае приходится привлекать к работе над проектом аналитика, который сможет сформулировать требования, выдвигаемые заказчиком и преобразовать их в задачи, которые требуется решить программистам. Также на этапе предварительного анализа начинается планирование разработки – осуществляется выбор технологии, производится выделение ресурсов. Некорректные требования, выдвигаемые заказчиком, проще всего изменить именно на этапе предварительного анализа. Если заказчик настаивает на выполнении своих требований, которые аналитик считает не реализуемыми, то следует отказаться от выполнения такого проекта. Пропуск этого этапа, скорее всего, повлечет в будущем необходимость изменения разработанного программного кода. Выбор неверной технологии приведет к невозможности окончания в срок разработки (например, если было принято решение реализовать web-приложение не с использованием технологий ASP.NET, PHP и т.п., а на C++; разрабатывать базу данных на низкоуровневом языке ассемблера и т.д.). Также на данном этапе требуется решить сколько разработчиков будут принимать участие в написание проекта. С одной стороны, нехватка разработчиков может привести к срыву сроков выполнение проекта и со стороны заказчика последуют штрафные санкции. С другой стороны, слишком большое количество

участников в проекте может привести к перерасходованию средств его бюджета.

На этапе разработки архитектуры осуществляется высокоуровневое проектирование, т.е. определяется какие будут созданы модули в системе, какие функции будут выполнять, как они будут взаимодействовать между собой. Правильно спроектированная архитектура является основой дальнейшей успешной работы над программным проектом. Она определяет каркас будущей разрабатываемой системы. Некорректно спроектированная архитектура системы (например, не учитывающая технические особенности ее аппаратной части или требования к нагрузке), либо вовсе не решит поставленной проблемы, либо приведет к ее некорректному решению. Основными задачами анализа является выделение сущностей (основных модулей), т.е. фактически решается из каких подсистем будет состоять проектируемая система. Если, например, одна из подсистем осуществляет работу с базой данных (БД), то следует выбрать способ для хранения и работы с данными (реляционная БД, XML, облако и т.д.). Следует также спроектировать модуль бизнес-логики, который будет отвечать за организацию взаимодействия компонентов друг с другом, а именно – какие входные данные получают модули, какие выдают выходные данные, каким образом следует структурировать данные обрабатываемые в модулях. Также обеспечивается информационная безопасность разрабатываемой системы (осуществляется разграничение прав пользователей, организуется конкурентный доступ к данным и т.д.). Если возможно, то на данном этапе решаются вопросы с повышением быстродействия разрабатываемой системы. Определяется также возможность масштабирования приложения, т.е. увеличения (или уменьшение) быстродействия системы в зависимости от возможностей предоставляемых ее аппаратной частью и возможность внедрения в нее новых модулей без существенного переписывания программного кода. Также определяется возможность реализуемости разработанной архитектуры системы и осуществляется первичное документирование системы с помощью набора бизнес-правил (например, при подаче на вход данных определенного типа на выходе должны формироваться следующие данные. При этом алгоритм осуществляющий такое преобразование на данном этапе во внимание не принимается). В завершении этого этапа следует заняться разработкой пользовательских интерфейсов (пока еще не в законченном варианте, а на уровне диаграмм или схем). На данном этапе необходимо убедится, что разработанная архитектура полностью обеспечивает решение всех пунктов технического задания.

На следующем этапе идет дополнительная проработка архитектуры, ее уточнение и детализация. Происходит движение от высокого абстрактного уровня архитектуры к низким уровням реальной системы. При использовании для разработки, например, объектно-ориентированной технологии программ-мирования осуществляется проектирование иерархии классов, осуществляется организация их взаимодействия, выбираются для использования в системе

определенные шаблоны проектирования и т.д. Производится решение некоторых задач (реализация отдельных модулей с целью определения их работоспособности) и на основе полученных результатов принимать решение о дальнейшем ходе выполнения работ.

На этапе кодирования и отладки осуществляется непосредственно написание программного кода с использованием выбранного языка программирования для реализации решения задач, поставленных на предыдущих этапах. Выполняется разработка хранилищ данных и пользовательских интерфейсов, осуществляется детальная реализация пунктов технического задания, производится первичное тестирование и исправление критических ошибок. Выполнение этого пункта представляет из себя итеративный процесс.

Готовый программный продукт подвергается тестированию. На данном этапе происходит проверка разработанной системы на соответствие требованиям технического задания, осуществляется поиск возможных ошибок как логики (например, проверка на некорректный ввод), так и интерфейсов (неверные формулировки, опечатки). Если в его ходе обнаруживаются ошибки, то происходит возврат к предыдущему этапу и осуществляется доработка функционала программы на основе выявленных недостатков.

На всех этапах разработки ПО выполняется документирование. Документирование на всех этапах разработки позволяет получить лучшее представление о работе разрабатываемой системы, служит руководством при проведении тестирования программы. Также документирование облегчает процесс дальнейшего сопровождения программы. При документировании программы необходимо также составить руководство по эксплуатации для ее пользователей.

После завершения тестирования происходит внедрение программы. Сюда входят процессы установки и наладки системы на стороне заказчика, включая финальное выявление ошибок и их исправление. Заканчиваться данный этап должен запуском системы и окончательной передачей ее заказчику. При разработке относительно несложных программных комплексов данный этап практически не требует затрат времени со стороны разработчиков.

После внедрения системы наступает этап сопровождения в течение ее жизненного цикла. На этом этапе по желанию заказчика можно ввести дополнительный функционал в систему. В целом этапы сопровождения аналогичны этапам разработки, но акцент в большей степени смещается в сторону написания программного кода.

На практике при реализации некритичных приложений часть перечисленных этапов может быть пропущена (например, выработка требований, тестирование, внедрение). В критичных или формальных приложениях будут присутствовать практически все перечисленные этапы разработки ПО. Следует отметить, что хотя написание программного кода и является основной работой при разработке ПО, но написание кода – это не

есть основа разрабатываемого приложения. Нельзя сказать, что приложение функционирует только за счет написания кода, верно, скорее обратное – приложение нормально работает за счет создания грамотной архитектуры на этапах, предваряющих этап кодирования. При плохо спроектированной архитектуре приложения даже хорошо написанный код, скорее всего, не сможет обеспечить его нормальное функционирование. В то время как при хорошо спроектированной архитектуре приложения плохо написанный код окажет меньшее негативное воздействие на его функционирование.

2. ОСНОВНЫЕ ПАРАДИГМЫ ПРОЦЕССА РАЗРАБОТКИ ПО

2.1. Каскадная модель разработки ПО

Основными парадигмами процесса разработки ПО считаются модель waterflow (каскадная или водопадная модель) и модель agile (гибкая модель разработки).

Каскадная модель— модель процесса разработки ПО, в которой процесс разработки выглядит как поток, последовательно проходящий фазы анализа требований, проектирования, реализации, тестирования, внедрения и сопровождения. Если все фазы разработки данной модели обозначить в виде прямоугольников и процесс перехода с одной фазы разработки на другую обозначать в виде направленных стрелок, то получится изображение напоминающие водопад (рис. 2.1), чем и обусловлено название данной модели.

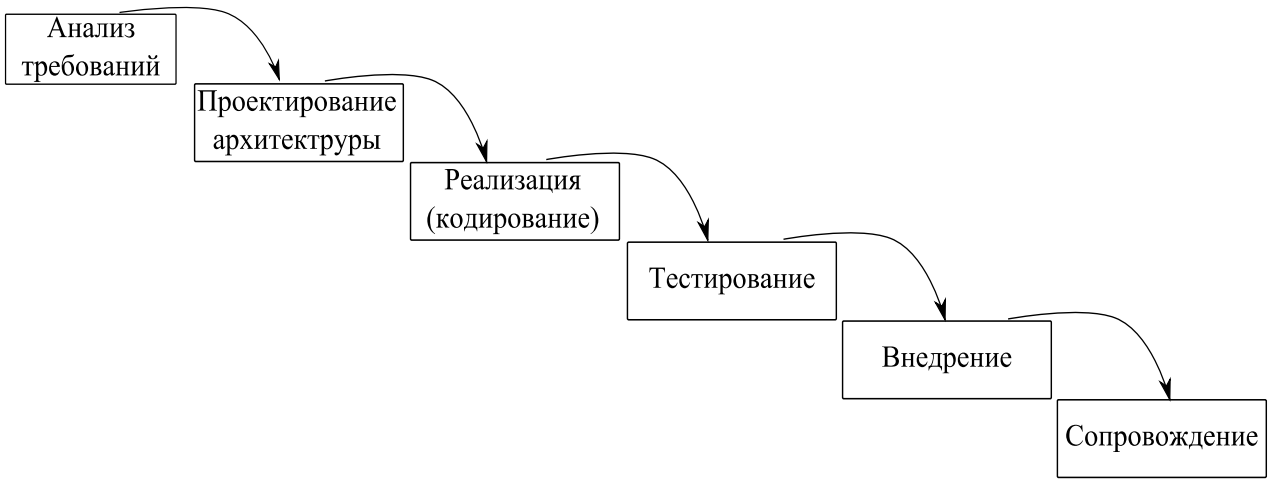

Рис. 2.1. Разработка ПО с использованием каскадной модели

Каскадная модель является старейшей парадигмой процесса разработки ПО [2]. В рамках каскадной модели, разработчики ПО переходят от одной стадии к другой строго последовательно. Этап анализа требований является началом работы над проектом. В результате выполнения этого этапа формулируется список требований к разрабатываемому ПО. После того как требования точно сформулированы, происходит переход к проектированию

архитектуры. В ходе выполнения данной фазы проекта создаются документы, детально описывающие для программистов план реализации и способ выполнения указанных требований. После полного завершения проектирования системы, разработчиками выполняется ее программная реализация. На этой же стадии процесса происходит интеграция отдельных компонентов, разрабатываемых различными командами программистов. Иногда интеграция компонентов выделяется в отдельный этап разработки. По завершении фазы реализации производится тестирование и отладка разработанного программного комплекса; на данном этапе устраняются все недостатки, которые проявились на предыдущих фазах разработки. На завершающих этапах разработанный программный комплекс внедряется и обеспечивается его сопровождение — внесение новой функциональности и устранение ошибок.

Тем самым, каскадная модель подразумевает, что переход от одной стадии разработки к другой происходит только после полного и успешного завершения предыдущего этапа, и что переходов назад, либо вперёд или перекрытия этапов — не происходит.

Однако, существуют модифицированные каскадные модели, имеющие небольшие или даже значительные отклонения от описанного процесса разработки.

Достоинством каскадной модели разработки ПО является, то что она дает план и временной график по всем стадиям проекта, что позволяет упорядочить ход разработки ПО. Недостатки каскадной модели следующие:

1) Работа над реальными проектами обычно требуют отклонения от стандартной последовательности выполнения шагов разработки.

2) Данная технология разработки базируется на точной постановке задач и формулирования всех исходных требований к ПО, хотя реально в начале проекта, как правило, требования заказчика определены лишь частично.

3) Результаты работы над таким проектом будут переданы заказчику только по его завершению. Хотя во время выполнения проекта у заказчика могут появится дополнительные требования к разрабатываемому программному комплексу. Таким образом, разработчики не смогут внести в этот проект какие-либо изменения в ходе его выполнения.

2.2. Итеративная модель разработки ПО

Итеративный подход в разработке ПО заключается в выполнении этапов разработки параллельно с непрерывным анализом полученных результатов и корректировкой предшествующих фаз работы. Проект при таком подходе в каждой фазе развития проходит повторяющийся цикл: *Планирование — Реализация — Проверка — Оценка* (англ. plan-do-check-act cycle) (рис. 2.2)

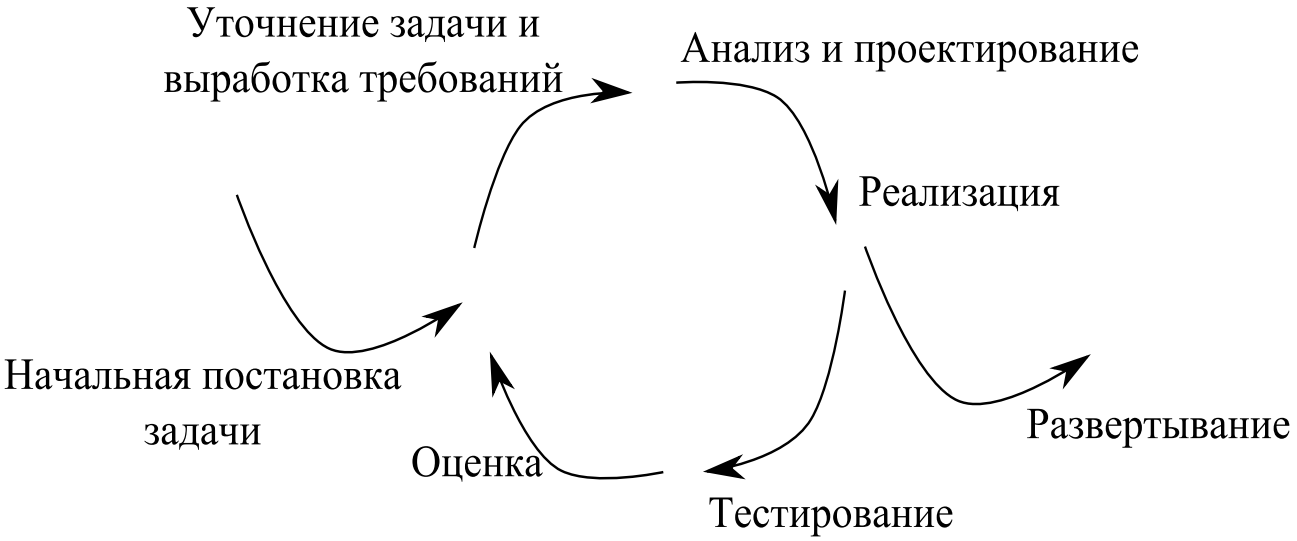

Рис. 2.2. Разработка ПО с использованием итеративной модели

Достоинствами данного подхода применительно к разработке ПО являются:

1) Уменьшение степени воздействия серьёзных рисков на ранних стадиях проекта, что в свою очередь означает снижение затрат на их устранение.

2) Организация эффективной обратной связи команды, работающей на программным проектом, с заказчиком и создание продукта, реально отвечающего его потребностям.

3) Сосредоточение усилий на решении наиболее критичных и определяющих задач в рамках проекта.

4) Постоянное итеративное тестирование, дающее возможность оценить успешность выполнения всего проекта в целом.

5) Раннее обнаружение конфликтов между требованиями, моделями и реализацией проекта.

6) Более равномерное распределение нагрузки между всеми участниками проекта.

7) Эффективное использование профессиональных компетенций и навыков, приобретенных участниками проекта.

8) Затраты распределяются по всему проекту, а не группируются в его конце.

Вместе с гибкостью и возможностью быстро реагировать на изменения, итеративные модели привносят дополнительные сложности в управление проектом и отслеживание его хода. При использовании итеративного подхода значительно сложнее становится адекватно оценить текущее состояние проекта и спланировать долгосрочное развитие событий, а также предсказать сроки и ресурсы, необходимые для обеспечения определенного качества результата.

Примером реализации итеративного подхода является методология разработки ПО RUP (Rational Unified Process), созданная компанией Rational Software.

2.3. Спиральная модель разработки ПО

Данная модель была впервые предложена в 1986 году Барри Боэмом. В рамках данной модели процесс разработки ПО представляется сочетанием проектирования и прототипирования на каждой фазе проекта с целью объединения преимуществ восходящей и нисходящей концепции разработки ПО. В данной модели особый акцент делается на начальные этапы жизненного цикла программного продукта: анализ и проектирование.

Другой отличительной особенностью данной модели является специальное внимание рискам, влияющим на организацию жизненного цикла. Создателем данной модели были сформулированы следующие десять наиболее распространённых (по приоритетам) рисков [3]:

1) Дефицит специалистов.
2) Нереалистичные сроки и бюджет.
3) Реализация несоответствующей функциональности.
4) Разработка неправильного пользовательского интерфейса.
5) Ненужная оптимизация оттачивания деталей.
6) Непрекращающийся поток изменений.
7) Нехватка информации о внешних компонентах, определяющих окружение системы или вовлечённых в интеграцию.
8) Недостатки в работах, выполняемых внешними (по отношению к проекту) ресурсами.
9) Недостаточная производительность получаемой системы.
10) «Разрыв» в квалификации специалистов разных областей знаний.

Подавляющие большинство перечисленных рисков связано с различными аспектами процесса организации и взаимодействия специалистов в проектной команде. Графическое представление процесса разработки ПО с использованием данной модели приведено на рис. 2.3.

Каждый виток спирали отвечает созданию фрагмента или версии программного обеспечения, на нем производится детализация целей и характеристик проекта, дается оценка его качеству и планируются работы следующего витка спирали. Т. е. на каждом витке спирали углубляются и последовательно конкретизируются детали проекта и в результате выбирается обоснованный вариант, который доводится до реализации. Любой виток спирали подразделяется на 4 сектора в которых производится:

1) Сравнение альтернатив, выделение рисков и определение способов борьбы с ними.
2) Определение задач, альтернатив и ограничений.
3) Разработка и верификация очередной части программного комплекса.
4) Планирование следующих итераций.

На любом витке спирали могут использоваться различные модели процесса разработки ПО. В результате, после выполнения перечисленных операций на нескольких витках спирали, на выходе получается готовый программный продукт. Данный подход совмещает в себе возможности каскадной модели и модели прототипирования. Применение итераций при

разработке позволяет отразить существующий спиральный цикл создания системы. В данной модели неполное окончание работ на каждом этапе позволяет переходить на следующий, не дожидаясь полного завершения работы на текущем этапе. При рассматриваемом способе разработки незавершенную работу предыдущих этапов можно будет выполнить в следующей итерации. Главная задача — как можно быстрее показать пользователям системы работоспособный продукт, тем самым активизируя процесс уточнения и дополнения требований. Основной проблемой спирального цикла является определение момента перехода на следующий этап. Для ее решения вводят временные ограничения на каждый из этапов жизненного цикла. Переход осуществляется в соответствии с этим временным графиком, даже если не вся запланированная работа была закончена. График выполнения работ составляется на основе статистических данных, полученных в предыдущих проектах, и личного опыта разработчиков.

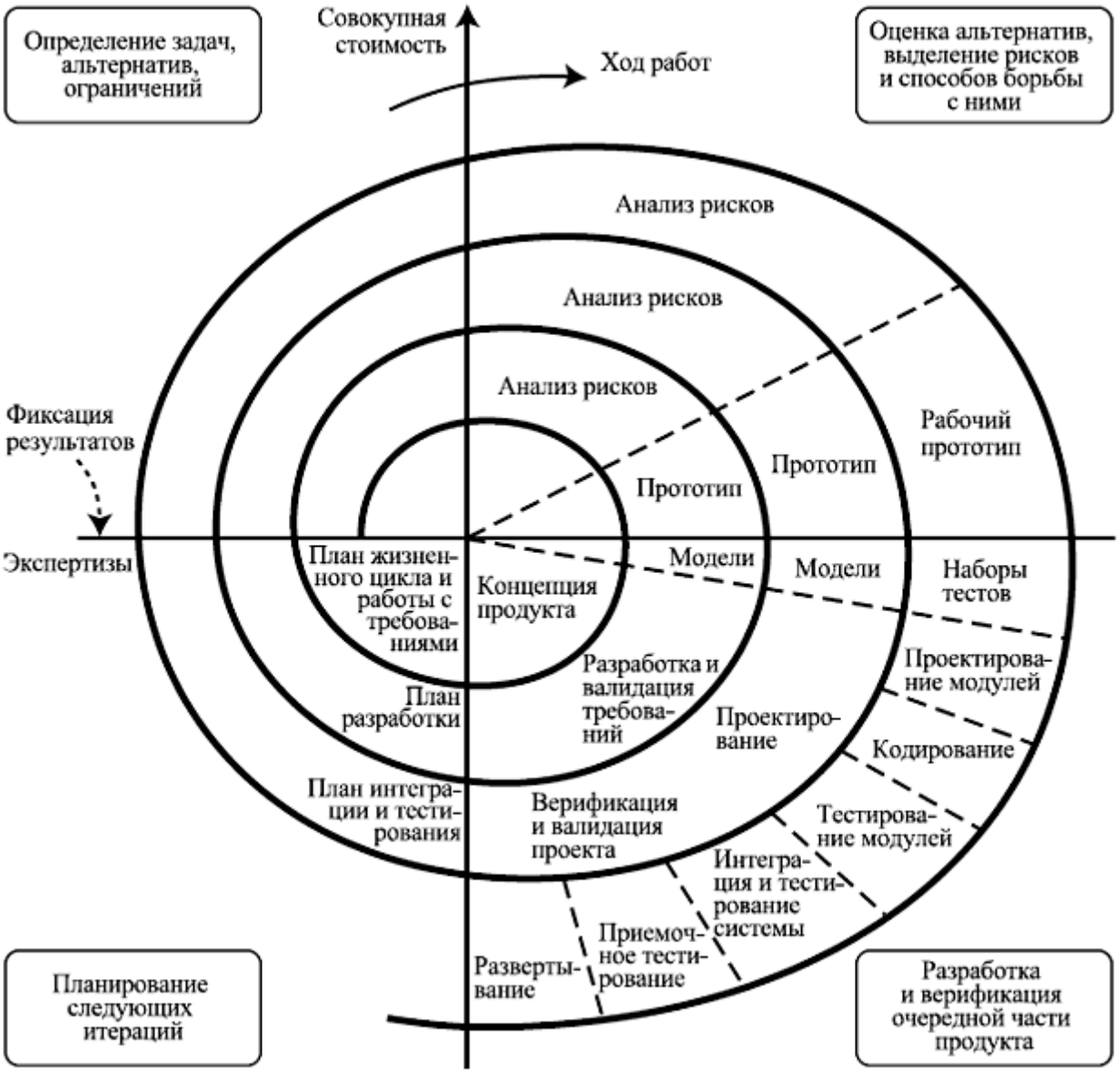

Рис. 2.3. Разработка ПО с использованием спиральной модели

Методология быстрой разработки приложений RAD (Rapid Application Development) является одной из возможных реализаций технологии разработки ПО, применяемой в рамках спиральной модели. Под термином RAD обычно понимается процесс разработки ПО, состоящий из трех основных частей:

1) Команды программистов, число участников в которой составляет от 2 до 10 человек.

2) Точно спланированного производственного графика работ, длительность которого составляет от 2 до 6 месяцев.

3) Повторяющегося цикла, при котором участники проекта, по мере того как в создаваемом приложении реализуют часть требуемых функций, запрашивают и реализуют в продукте требования, полученные в результате общения с заказчиком.

В методологии RAD жизненный цикл ПО состоит из четырёх фаз:

1) Фаза определения требований и анализа.

2) Фаза проектирования.

3) Фаза реализации.

4) Фаза внедрения.

Спиральная модель разработки в основном ориентирована на большие, дорогостоящие и сложные проекты.

2.4. V модель разработки ПО

V-Model (или VEE модель) является моделью разработки ПО, нацеленной на упрощение понимания сложностей, возникающих в связи с разработкой таких программных комплексов. Иллюстрация разработки ПО с использованием данной модели приведена на рис.5. Отличительной чертой V-образной модели является то, что детализация проекта возрастает при движении слева направо, одновременно с течением времени, и ни то (детализацию проекта), ни другое (течение времени) нельзя повернуть вспять.

В рамках данной модели в проекте итерации производятся по горизонтали, между левой и правой сторонами буквы V (рис. 2.4). V-модель является разновидностью каскадной модели, в ней задачи, связанные с разработкой идут сверху вниз по левой стороне буквы V, а задачи, связанные с тестированием — вверх по правой стороне буквы V. Внутри буквы V проводятся горизонтальные линии, отражающие, какое влияние результаты каждого из этапа разработки оказывают на разработку и развитие системы тестирования в целом на каждой из фаз проекта. Данная модель основывается на том, что испытания, проводимые перед вводом системы в эксплуатацию, базируются, в большей своей части, на требованиях предъявляемых к системе; системное тестирование — на требованиях и архитектуре разрабатываемого программного комплекса; комплексное тестирование — на требованиях, архитектуре и интерфейсах; а компонентное тестирование — на требованиях, архитектуре, интерфейсах, а также алгоритмах, реализованных в разрабатываемом комплексе.

Предназначением V-модели является обеспечение поддержки выполнения хода планирования и реализации проекта. Использование V- образной модели подразумевает решение следующих задач в ходе работы над проектом:

1) Минимизация рисков. Применение V-образной модели делает проект более легко управляемым и повышает качество контроля работ, которые ведутся в рамках реализации данного проекта путём стандартизации промежуточных целей и описания соответствующих им результатов, а также лиц, ответственных за достижения этих целей. Все это позволяет выявить отклонения в ходе работы над проектом и риски на ранних стадиях и, как следствие, повышает качество управления работы над проектами, уменьшая при этом риски.

Рис. 2.4. Разработка ПО с использованием V-модели

2) Повышение и гарантии качества разрабатываемого ПО. V-Model представляет из себя стандартизованную модель разработки, что позволяет добиться при реализации проекта результатов желаемого качества. Промежуточные результаты, в рамках данной модели, могут быть проверены на ранних стадиях. Применяемое универсальное документирование облегчает читабельность и верифицируемость.

3) Уменьшение общей стоимости проекта**.** Стоимость ресурсов, которые требуются на разработку, производство, управление и поддержку программного комплекса, может быть заранее просчитана и проконтролирована. Результаты, достигнутые в ходе реализации проекта, также легко прогнозируются. Это позволяет уменьшить затраты на последующие стадии и проекты.

4) Повышение качества коммуникации между участниками проекта. Универсальное описание всех элементов и условий способствует установле-

нию взаимопонимания между всеми участниками проекта. Все это приводит к более полному пониманию между пользователем, покупателем, поставщиком и разработчиком.

Достоинства данной модели разработки ПО заключаются в следующем [4]:

1) Пользователи V-Model участвуют в разработке и поддержке V-модели. Комитет по контролю за изменениями поддерживает проект и собирается раз в год для обработки всех полученных запросов на внесение изменений в V-Model.

2) На старте любого проекта V-образная модель может быть адаптирована под этот проект, так как эта модель не зависит от типов организаций и проектов.

3) V-model позволяет разбить деятельность на отдельные шаги, каждый из которых будет включать в себя необходимые для него действия, инструкции к ним, рекомендации и подробное объяснение деятельности.

4) В модели особое значение придается планированию, направленному на верификацию и аттестацию разрабатываемого продукта на ранних стадиях его разработки. Фаза модульного тестирования подтверждает правильность детализированного проектирования. Фазы интеграции и тестирования реализуют архитектурное проектирование или проектирование на высшем уровне. Фаза тестирования системы подтверждает правильность выполнения этапа требований к продукту и его спецификации.

5) В модели предусмотрены аттестация и верификация всех внешних и внутренних полученных данных, а не только самого программного продукта.

6) В V-образной модели определение требований выполняется перед разработкой проекта системы, а проектирование ПО — перед разработкой компонентов.

7) Модель определяет продукты, которые должны быть получены в результате процесса разработки, причем каждые полученные данные должны подвергаться тестированию.

8) Благодаря модели менеджеры проекта могут отслеживать ход процесса разработки, так как в данном случае вполне возможно воспользоваться временной шкалой, а завершение каждой фазы является контрольной точкой.

Недостатками V-модели являются [4]:

1) Модель не предусматривает работу с параллельными событиями.

2) В модели не предусмотрено внесение требования динамических изменений на разных этапах жизненного цикла.

3) Тестирование требований в жизненном цикле происходит слишком поздно, вследствие чего невозможно внести изменения, не повлияв при этом на график выполнения проекта.

4) В модель не входят действия, направленные на анализ рисков.

5) Некоторый результат можно посмотреть только при достижении низа буквы V.

3. МЕТОДОЛОГИИ ПРОГРАММИРОВАНИЯ

Среди существующих в настоящее время методологий программирования можно выделить следующие виды:

1) Agile и ее разновидности (XP - eXtreme Programming, Lean, Scrum , FDD - Feature driven development и др.)
2) RAD (Rapid Application Development).
3) RUP (Rational Unified Process).
4) MSF (Microsoft Solutions Framework).
5) DSDM (Dynamic Systems Development Method).
6) TDD (Test-Driven Development).

3.1. Agile методологии программирования

Гибкая методология разработки (англ. Agile software development, agile-методы) — представляет из себя серию подходов к разработке ПО, ориентированных, прежде всего, на использование итеративной разработки, формирование требований в ходе работы над проектом и их реализацию в результате постоянного взаимодействия внутри самоорганизующихся рабочих групп, которые состоят из специалистов разного профиля.

Большинство из гибких методологий нацелены на минимизацию рисков путём приведения процесса разработки ПО к серии коротких циклов, называемых итерациями, продолжительность которых обычно составляет две-три недели. Любая из таких итераций представляет собой программный проект в миниатюре и включает в себя задачи, необходимые для выдачи мини-прироста по функциональности разрабатываемого приложения, а именно: анализ требований, планирование, проектирование, программмирование, тестирование и документирование. Хотя достигнутых результатов в рамках одной итерации, как правило, недостаточно для выпуска новой версии программного продукта, подразумевается, что гибкий программный проект готов к выпуску в конце каждой итерации. При завершении каждой итерации команда, работающая над проектом, выполняет переоценку приоритетов разработки.

Agile-методы основной акцент сосредотачивают на непосредственном общении между участниками команды, работающей над реализацией проекта. Подавляющее большинство agile-команд располагаются в одном офисе. Такая команда должна, как минимум, включать и разработчиков и «заказчиков» (заказчик или его полномочный представитель, определяющий требования к продукту; эту роль может выполнять менеджер проекта, бизнес-аналитик или клиент). В состав этой команды также могут входить: тестировщики, дизайнеры интерфейса, технические писатели и менеджеры.

Основной единицей измерения, используемой в рамках agile-методов, является рабочий продукт. Поскольку при применении agile-методов для разработки отдают предпочтении непосредственному общению, то это позволяет резко, по сравнению с другими методологиями разработки ПО,

уменьшить объём письменной документации, регламентирующей как процесс разработки, так и требования, предъявляемые к разрабатываемой системе. Данная особенность agile-методов является одним из поводов для их критики (в рамках которой эти методы разработки характеризуются как недисциплинированные).

Agile методология представляет из себя семейство процессов разработки, а не единственный подход в разработке ПО, и ее основные положения определяются Agile Manifesto. Agile методология не включает практик, а формулирует ценности и принципы, которыми должны руководствоваться команды, работающие над реализацией программного проекта.

Основные идеи, сформулированные Agile Manifesto [5]:

1) Люди и взаимодействие важнее процессов и инструментов.

2) Работающий продукт важнее исчерпывающей документации.

3) Сотрудничество с заказчиком важнее согласования условий контракта.

4) Готовность к изменениям важнее следования первоначальному плану.

Принципы, которые разъясняет Agile Manifesto [5]:

1) Удовлетворение клиента за счёт ранней и бесперебойной поставки ценного ПО;

2) Приветствие изменений требований даже в конце разработки (это может повысить конкурентоспособность полученного продукта);

3) Частая поставка рабочего ПО (каждый месяц или неделю или ещё чаще);

4) Тесное, ежедневное общение заказчика с разработчиками на протяжении всего проекта;

5) Проектом занимаются мотивированные личности, которые обеспечены нужными условиями работы, поддержкой и доверием;

6) Рекомендуемый метод передачи информации — личный разговор (лицом к лицу);

7) Работающее ПО — лучший измеритель прогресса.

8) Спонсоры, разработчики и пользователи должны иметь возможность поддерживать постоянный темп на неопределённый срок;

9) Постоянное внимание улучшению технического мастерства и удобному дизайну;

10) Простота — искусство не делать лишней работы;

11) Лучшие технические требования, дизайн и архитектура получаются у самоорганизованной команды;

12) Постоянная адаптация к изменяющимся обстоятельствам.

Один из пунктов критики при agile-подходе часто пренебрегают созданием плана («дорожной карты») развития программного продукта, а также управлением требованиями, в процессе которого и синтезируется данная «карта». Использование данной методологии не подразумевает далеко идущих планов по управлению требованиями (по сути, управления требованиями просто не существует в данной методологии), а подразумевает возможность заказчика вдруг и неожиданно в конце каждой итерации

выставлять новые требования, часто противоречащие архитектуре уже созданного и поставляемого продукта. Данное обстоятельство может приводить к катастрофическим «авралам» с массовым рефакторингом и переделками практически на каждой очередной итерации.

Также, считается, что работа в стиле agile мотивирует разработчиков решать все поступившие им задачи простейшим и быстрейшим возможным способом, при этом они зачастую не обращают внимания на правильность кода с точки зрения требований нижележащей платформы (подход — «работает, и ладно», при этом не учитывается, что может перестать работать при малейшем изменении или же дать тяжёлые к воспроизводству дефекты после реального развёртывания у клиента). Это приводит к снижению качества разработанного продукта и накоплению дефектов.

3.1.1. Экстремальное программирование

Двенадцать основных приёмов экстремального программирования [6, 7] могут быть объединены в четыре группы:

1) Короткий цикл обратной связи (Fine scale feedback)
   1.1. Разработка через тестирование (Test driven development)
   1.2. Игра в планирование (Planning game)
   1.3. Заказчик всегда рядом (Whole team, Onsite customer)
   1.4. Парное программирование (Pair programming)
2) Непрерывный, а не пакетный процесс
   2.1. Непрерывная интеграция (Continuous Integration)
   2.2. Рефакторинг (Design Improvement, Refactor)
   2.3. Частые небольшие релизы (Small Releases)
3) Понимание, разделяемое всеми
   3.1. Простота (Simple design)
   3.2. Метафора системы (System metaphor)
   3.3. Коллективное владение кодом (Collective code ownership) или выбранными шаблонами проектирования (Collective patterns ownership)
   3.4. Стандарт кодирования (Coding standard or Coding conventions)
4) Социальная защищенность программиста (Programmer welfare): 40-часовая рабочая неделя (Sustainable pace, Forty hour week)

Использование данной методологии предусматривает создание автоматических тестов (программный код, который написан специально для тестирования логики другого программного кода). Наибольшее внимание уделяется двум вариантам тестирования:

1. Тестирование модулей (unit testing).
2. Функциональное тестирование.

При использовании данного подхода программист не может быть уверен в правильности написанного им кода до того момента времени, пока созданный код успешно не пройдет все модульные тесты модулей разрабатываемой им системы. Модульные тесты (тесты модулей, юнит-тесты) позволяют разработчикам, реализующим программный код проекта,

убедиться в том, что каждый из них по отдельности пишет корректно работающий код. Модульные тесты также помогают другим разработчикам понять, зачем нужен тот или иной фрагмент программы и как он функционирует. При изучении кода тестов модулей логика работы самого тестируемого кода становится более понятной, т.к. видно, как он должен функционировать. Тесты модулей также облегчают разработчику проведение рефакторинга кода.

Функциональные тесты используются для проверки функционирования логики, образуемой взаимодействием нескольких (достаточно часто - довольно большого объема) частей. Эти тесты менее детальны, чем модульные тесты, но покрывают значительно больший объем программного кода, естественно, что при этом шанс обнаружить какое-либо некорректное поведение разрабатываемой системы будет значительно больше. Из-за этого обстоятельства при разработке больших коммерческих проектов создание функциональных тестов часто имеет боле высокий приоритет, чем написание юнит-тестов.

В методологии XP наиболее популярным является подход, получивший название TDD (Test Driven Development). При использовании данного подхода в начале работы над проектом пишется тест, который изначально не проходит (т.к. логики, которую он должен проверять, ещё просто не существует), и лишь затем реализуется программная логика, необходимая для того, чтобы этот тест прошел. В определенной степени TDD позволяет писать код, который более удобен в использовании, так как при создании теста, когда логики функционирования программы ещё нет, наиболее просто принять меры, обеспечивающее удобство работы с будущей системой.

Главной целью «игры в планирование» является быстрое формирование приблизительного плана работ и постоянное его обновление по мере детализации и конкретизации условий решаемой задачи. При реализации «игры в планирование» используется набор бумажных карточек, на них записываются пожелания и требования заказчика (customer stories), а также приблизительный план работы по выпуску следующих одной или нескольких версий (отличающихся незначительными изменениями от уже частично функционирующих) разрабатываемого продукта. Основополагающим фактором, благодаря которому применение данной методологии оказывается эффективным, является то, что в такой ситуации заказчик отвечает за принятие бизнес-решений, а команда разработчиков отвечает за принятие и реализацию технических решений. Если не выполняется данное условие, то весь процесс распадается на части.

«Заказчик» в методологии XP выступает конечный пользователь программного продукта, а не лицо, производящее оплату счетов. В рамках методологии XP утверждается, что заказчик должен быть всё время на связи и доступен для вопросов.

Прием «парное программирование» предполагает, что весь код пишется парами программистов, которые работают за одним компьютером. Один из

этих программистов работает непосредственно с текстом исходного кода программы, а другой просматривает его работу, а также следит за общей картиной происходящего. При необходимости клавиатура свободно передается от одного программиста к другому. В течение работы над проектом пары не фиксированы: рекомендуется даже их перемешивать, чтобы каждый программист в команде мог получить хорошее представление о всей разрабатываемой системе. Использование парного программирования, по мнению создателей данной методологии, усиливает взаимодействие внутри команды.

Если интеграция частей разрабатываемой системы выполняется достаточно часто, то в этом случае можно избежать значительной части связанных с этим проблем. В традиционных методологиях интеграция обычно выполняется в самом конце работы над программным комплексом, когда считается, что все составные части разрабатываемой системы уже полностью готовы. В рамках методологии XP интеграция программного кода всей системы выполняется несколько раз в день, после того, как разработчики убедились в том, что все тесты модулей корректно срабатывают.

Рефакторингом называется методика улучшения кода без изменения его функциональности. В методологии XP предполагается, что однажды созданный программный код, в ходе работы над проектом почти наверняка будет переделан несколько раз. Разработчики, использующие методологию XP, изменяют созданный ранее программный код с целью его улучшения. Отсутствие юнит-тестов, покрывающих объем разработанного исходного кода, может спровоцировать отказ от выполнения рефакторинга в связи с опасениями поломать систему, что в конечном итоге приводит к постепенной деградации кода.

Версии (releases) разрабатываемого программного продукта должны поступать в эксплуатацию как можно чаще. Работа, которая выполняется для реализации очередной версии, должна занимать как можно меньше времени. При этом каждая очередная версия, разрабатываемого программного продукта, должна быть достаточно осмысленной с точки зрения полезности для бизнеса.

Чем раньше произойдет выпуск первой рабочей версии продукта, тем раньше заказчик начнет получать (при оптимистическом сценарии развития событий) за счёт неё дополнительную прибыль. В любом случае, чем раньше заказчик введет в эксплуатацию разработанный программный продукт, тем скорее разработчики получат от него информацию о том, что соответствует (или не соответствует) требованиям заказчика в разработанной системе. Эта информация, с высокой долей вероятности, окажется весьма ценной при планировании следующего выпуска.

Разработчики, использующие методологию XP, исходят из того, что в ходе работы над реализацией проекта условия задачи могут неоднократно поменяться. Следовательно, разрабатываемый программный продукт не следует проектировать целиком и полностью заранее. В самом начале работы

над программной реализацией разрабатываемой системы попытка от начала и до конца ее детально спроектировать будет приводить к лишним затратам времени. Методология XP предполагает, что проектирование представляет из себя достаточно важный процесс, так что его следует постоянно выполнять в ходе всего процесса работы над проектом. Проектирование должно выполняться небольшими этапами, принимая во внимание постоянно изменяющиеся требования со стороны заказчика. При разработке проекта на любом из его промежуточных этапов следует использовать наиболее простой дизайн, при помощи которого можно обеспечить решение текущей задачи. При этом следует модифицировать его по мере изменений условий задачи.

В рамках методологии XP метафора системы (system metaphor) является аналогом того, что в большинстве других методик называется архитектурой. Архитектурой называется представление о компонентах системы и их взаимосвязях между собой. Анализ архитектуры ПО необходимо проводить с целью выяснения мест в системе, нуждающихся в добавлении новой функциональности, а также выяснения того, с какими частями разрабатываемой системы будет взаимодействовать вновь добавляемый компонент.

Метафора системы предоставляет участникам проекта информацию о том, каким образом система функционирует в текущий момент времени, в каких ее местах происходит добавление новых частей и какую форму должны принять вновь добавляемые компоненты.

Выбор адекватной метафоры позволяет участникам проекта улучшить понимание того, каким образом будет функционировать разрабатываемая ими система. В большинстве случаев такой выбор представляет из себя достаточно не тривиальную задачу.

Все участники команды в процессе работы над проектом должны строго придерживаться требований общих стандартов кодирования. Это приводит к тому, что, во-первых, участники команды не будут тратить время на дискуссии между собой о предметах, которые практически не оказывают никакого влияния на затраты времени необходимые для реализации проекта, а во-вторых, в процессе работы над проектом, обеспечивается достаточное эффективное исполнение и остальных практик.

Если участники, работающие над проектом, не используют единые стандарты кодирования, то разработчикам будет достаточно сложно выполнять рефакторинг; при обмене партнеров между парами будет возникать больше затруднений; в общем и целом, ход работы по реализации проекта будет замедляться. В рамках использования методологии XP требуется стремиться к тому, чтобы по исходному тексту программы невозможно было определить авторство того или иного участка программного кода. Все участники проекта должны работать унифицированно, создавая впечатление, что программный код пишется одним человеком. Участники проекта перед началом работы над ним должны сформулировать определенный набор правил, после чего все они должны будут их придерживаться в ходе

написания программного кода. Перечень сформулированных правил не должен быть слишком объемным, он также не должен регламентировать все мельчайшие детали процесса разработки. Необходимо просто дать общие указания по разработке, при помощи которых создаваемый программный код будет достаточно ясным для понимания любому из участников работы над проектом. В первоначальной стадии работы над проектом стандарт кодирования следует сделать более простым, а по мере выполнения работ по реализации проекта этот стандарт может постепенно усложняться в связи с накоплением опыта участниками проекта. На предварительную разработку стандарта в методологии XP не следует допускать слишком больших временных затрат.

Коллективное владение подразумевает, что любой участник работы над реализацией проекта отвечает за весь исходный код текста программы. Из этого положения вытекает, что любой из участников разработки имеет право модифицировать любой фрагмент создаваемой программы. Парное программирование представляет собой средство поддержки данной практики: программируя в разных парах, все участники знакомятся со всеми фрагментами исходного кода создаваемой системы. С одной стороны, принцип коллективного владения кодом позволяет ускорить выполнение процесса разработки (при возникновении ошибки ее может ликвидировать любой из участников команды). С другой стороны предоставление любому из участников команды права на модификацию кода создает дополнительные риски возникновения ошибок, допускаемых программистами, которые пишут исходный текст, не принимая во внимание наличие некоторых зависимостей между различными частями создаваемой системы. Однако правильно созданные модульные тесты являются решением данной проблемы. Если неучтенные программистом зависимости между различными частями системы будут приводить к возникновению ошибок, то последующий запуск модульных тестов будет неудачным.

3.1.2. Методология Scrum

Scrum (от англ. *scrum* «схватка») называется методология управления проектами, которая активно используется при создании ПО в рамках модели гибкой разработки. В методологии Scrum основное внимание заостряется на качественном управлении ходом работ по реализации проекта. Данная методология, помимо непосредственного использования в процессе разработки ПО, также может применяться и службой поддержки ПО.

Скрам (Scrum) представляет из себя набор правил, на основе которых выполняются работы по реализации проекта. Следование этим правилам позволяет в небольшие, жестко фиксированные по времени итерации (спринты) предоставлять заказчику функционирующее ПО, в котором реализованы новые возможности по сравнению с предыдущей версией с новыми возможностями. В начале выполнения работ над проектом в рамках спринта реализуются возможности, которые обладают наивысшим

приоритетом с точки зрения заказчика. Реализуемость добавляемого функционала к разрабатываемой системе в течение очередного спринта определяется участниками команды в его начале, на стадии планирования работ (в рамках одного спринта). Обозначенные в начале очередного спринта цели, которые должны быть достигнуты к моменту его окончания, не подлежат корректировке. Поскольку спринт представляет строго фиксированный промежуток времени и при этом его длительность невелика, то это позволяет придать ходу работ по реализации проекта гибкость и предсказуемость.

По завершению очередного спринта должен происходить рост функционала разрабатываемой системы. Продолжительность отдельного спринта обычно составляет от 2 до 4 недель. Иногда продолжительность спринта может доходить до 6 недель (стандарт Scrum разработки, принятый в фирме Nokia). Однако, большинство разработчиков, использующих данную методологию в своей работе, придерживаются мнения, что более короткий спринт позволяет более гибко управлять ходом разработки. В этом случае версии разрабатываемого ПО выходят более часто, как следствие сокращается интервал времени, после которого приходят отзывы потребителей, эксплуатирующих разрабатываемую программу, что в свою очередь снизить расход времени на реализацию ненужных вариантов.

Однако более длительные спринты могут предоставить участникам работы над проектом ряд преимуществ. В этом случае появляется резерв времени, который можно израсходовать на решение проблем, возникающих в ходе реализации проекта. Владелец проекта в этом случае может сократить издержки времени на показ версий разрабатываемой системы, обсуждения и т. п. В целом, выбор длительности спринта возлагается на команду, реализующую проект. При выборе длительности спринта команда должна учитывать особенности предметной области, для которой реализуется проект, опыт разработки своих участников, требования, выдвигаемые к разрабатываемой системе. Выбор оптимальной длительности спринта приходит с накоплением опыта. Предварительная оценка, которая измеряется в очках истории, может быть использована для характеристики трудозатрат по выполнению работ в рамках одного спринта. Она должна быть зафиксирована в резерве проекта.

Резерв проекта (product backlog) представляет из себя перечень требований к функциям, которые должны реализовываться в разрабатываемой системе. Этот список требований упорядочивается в соответствии с их приоритетами, задачи находящиеся в его начале должны быть реализованы в первую очередь. Элементы перечня требований называются «пожеланиями пользователя» (user story) или элементами резерва (backlog items). Все участники команды могут редактировать резерв проекта.

В резерв спринта (sprint backlog) включается функциональность, которую владелец проекта выбирает из резерва проекта. Все подлежащее реализации функции должны быть разбиты на задачи, трудоемкость решения

подлежит оценке участников скрам-команды. В процессе работы ежедневно участники работы над проектом оценивают объем работ, который необходимо выполнить для окончания спринта.

Текущий ход выполнения работ по реализации проекта при применении данной методологии отображает диаграмма сгорания задач. Существуют разные виды диаграмм сгорания задач, но в основном используются только два:

1) Диаграмма сгорания работ для спринта отображает, какое количество поставленных задач успешно решено в текущем спринте и какие еще задачи требуется решить до его окончания.

2) Диаграмма сгорания работ для выпуска отображает, какое количество поставленных задач успешно решено и какие из них еще только требуется решить до выпуска продукта (для ее построения обычно используется информация, полученная в ходе выполнения нескольких спринтов).

Функции, запланированные к реализации и добавленные в резерв спринта, часто называют историей. Обычно история строится по следующему шаблону: «Будучи пользователем <тип пользователя> я выполняю действие <вид действия>, с целью получения следующего результата <результат>». Данный шаблон весьма удобен для понимания и разработчикам проекта и его заказчикам.

Завершение спринта раньше срока его планируемого окончания возможно лишь в исключительных случаях. Спринт останавливается, если участники команды понимают, что не могут достигнуть поставленных в его начале целей в отведенный промежуток времени. Владелец проекта также может остановить спринт, если достижение целей поставленных в его начале перестает быть актуальным. В случае остановки спринта участники работы над проектом обсуждают причины, приведшие к возникновению данной ситуации. Затем стартует новый спринт.

Участники скрам-команды обычно выполняют оценку сложности истории при помощи абстрактной метрики, не учитывающей затраты времени на написание программного кода в человекочасах. Обычно используется одна из следующих шкал: ряд Фибоначчи (1,2,3,5,8,13,21,34,55); линейную шкалу (1,2,3,4 … n); степень двойки (1,2,4,8 … 2^n). Задачи истории спринта (sprint story tasks) добавляются к историям спринта. Они служат для оценки временных затрат на решение поставленных задач в часах. Затраты времени на работу над любой из поставленных задач не должны превышать 12 часов (обычно участники команды решают, что продолжительность времени по реализации одной задачи должна равняться одному рабочему дню). Критерии, определяющие степень готовности разработанной части программного комплекса (DoD – Definition of Done), выбираются из журнала пожеланий пользователя. Скорость работы команды - это общее количество очков, набранных ею в ходе выполнения предыдущего спринта. Данная единица измерения помогает участникам команды получить представление, сколько историй они смогут реализовать при выполнении одного спринта.

Использование методологии Scrum для создания ПО подразумевает, что все участники заинтересованные в его успешном окончании играют при его реализации определенные роли, которые разбиваются на две группы. В первую группу входят участники, которые полностью включены в процесс реализации проекта и которые играют основные роли в этом процессе. В рамках данной группы участники играют следующие роли (основные роли):

1) Скрам-мастер (Scrum Master). В его обязанности входит проведение совещаний (Scrum meetings), контроль за соблюдением участниками команды всех принципов методологии Scrum. Он также является арбитром при возникновении противоречий между участниками команды и не допускает влияния на них отвлекающих факторов. Эта роль предназначена только для корректного выполнения скрам-процесса. Руководитель проекта не может выполнять функции скрам-мастера, поскольку скорее он играет роль владельца продукта.

2) Владелец продукта (Product Owner). В его обязанности входит представление интересов конечных пользователей и других лиц, заинтересованных в появлении разрабатываемой системы.

3) Скрам-команда (Scrum Team) — кросс-функциональная команда разработчиков проекта, участниками которой являются специалисты разных профилей: тестировщики, архитекторы, аналитики, программисты и т. д. Число участников команды в идеальном случае должно составлять 7±2 человека. Данная роль предполагает полное вовлечение в процесс разработки, причем вся команда отвечает за достигнутые (или не достигнутые) результаты как единое целое. В течение выполнения спринта только команда может вмешиваться в процесс разработки.

Все перечисленные выше категории участников создают программный продукт и заинтересованы в успешном завершении работ по его реализации. Вторую группу составляют участники, которые хотя и заинтересованы в удачном завершении проекта, но не так сильно, как участники первой группы. Во второй группе участники играют следующие роли (дополнительные роли):

1) Пользователи (Users).

2) Клиенты, продавцы (Stakeholders). Под ними подразумеваются лица, которые инициируют работы по реализации проекта и для которых его успешная реализация будет приносить выгоду. Они вовлекаются в скрам процесс только во время обзорного совещания по спринту (Sprint Review).

3) Управляющие (Managers) — люди, ответственные за управление персоналом.

4) Эксперты-консультанты (Consulting Experts)

При использовании методологии Scrum для разработки ПО используются такие артефакты, как обязательные (табл. 3.1) и дополнительные поля (табл. 3.2).

Таблица 3.1

Обязательные поля, используемые в методологии Scrum

Имя поля	Описание поля
ID	Уникальный идентификатор, порядковый номер, применяемый для идентификации историй в случае их переименования.
Название (Name)	Краткое описание истории. Оно должно быть однозначным, чтобы и разработчики, и владелец проекта могли понять, о чем идёт речь и отличить одну историю от другой.
Важность (Importance)	Степень важности данной истории, по мнению владельца проекта. Обычно представляет собой натуральное число, иногда для этой цели используются числа Фибоначчи. Чем больше значение, тем выше приоритет.
Предварительная оценка (initial estimate)	Начальная оценка объёма работ, необходимого для реализации истории по сравнению с другими историями. Измеряется в story point'ах. Приблизительно соответствует числу «идеальных человеко-часов».
Способ демонстрации (how to demo)	Краткое пояснение того, как завершённая задача будет продемонстрирована в конце спринта. Данное поле может представлять собой код автоматизированного теста для приёмо-сдаточного испытания.

Таблица 3.2

Дополнительные поля, используемые в методологии Scrum

Имя поля	Описание поля
Категория (track)	Например, «панель управления» или «оптимизация». При помощи этого поля владелец проекта может легко выбрать все пункты категории «оптимизация» и установить им низкий приоритет.
Компоненты (Components)	Указывает, какие компоненты (например, база данных, сервер, клиент) будут затронуты при реализации истории. Данное поле состоит из группы флажков, которые отмечаются, если соответствующие компоненты требуют изменений.
Инициатор запроса (requestor)	Владелец продукта может захотеть хранить информацию о всех заказчиках, заинтересованных в данной задаче. Это нужно для того, чтобы держать их в курсе дела о ходе выполнения работ.
ID в системе учёта дефектов (bug tracking ID)	Если при разработке используется отдельная система отслеживания ошибок, тогда в описании истории полезно хранить ссылки на все дефекты, которые к ней относятся.

В начале каждой новой итерации спринта происходит встреча участников разработки с целью его планирования. На этой встрече выполняются следующие действия:

1) Из резерва проекта выбираются задачи, которые должна решить за спринт-команда.

2) На основе поставленных перед командой задач создается резерв спринта. Оценка трудоемкости выполнения каждой задачи производится в идеальных человеко-часах.

3) Процесс решения задачи не должен превышать 12 часов или один день. Если поставленная задача слишком сложна, то ее следует разбить на ряд более мелких, функционально лучше управляемых подзадач.

4) Производится обсуждение способа реализации поставленных задач.

5) Максимум времени, отводимый на данную встречу, лежит в пределах от 4 до 8 часов. Длительность этого совещания определяется продолжительностью итерации, опытом участников команды и множеством других факторов.

В первой части этой встречи принимает участие как владелец проекта, так и скрам-команда. Целью этой части совещания является выбор задач, решение которых должно быть реализовано в виде программного кода, из резерва проекта. Во второй части встречи в обсуждении принимают участие только члены команды, которые наполняют резерв спринта и выбирают способ реализации решения поставленных задач.

Во время работы над проектом по методологии Scrum проводятся ежедневные совещания в одно и то же время. На этих совещаниях в качестве наблюдателей могут присутствовать участники второй группы, но обсуждение и принятие решений по ходу выполнения проекта возлагается только на участников первой группы. Длительность такого совещания не превышает 15 минут, и в течение одного спринта оно проводится в одном и том же месте. В течение ежедневного совещания каждый из участников отвечает на три вопроса:

1) Что удалось реализовать после окончания предыдущего ежедневного совещания?

2) Что будет реализовано к следующему ежедневному совещанию?

3) Какие затруднения препятствуют достижению поставленных целей спринта? (Решение этих проблем является прерогативой скрам-мастер. Обычно они решаются за рамками ежедневного совещания в составе лиц, работа которых непосредственно затрагивается данным препятствием).

Если над реализацией проекта работает несколько команд, то после ежедневного скрам-совещания проводится еще одно совещание, называемое «Скрам над скрамом». На нем несколько скрам-команд обсуждают работу, концентрируя свое внимание на общих областях и взаимной интеграции. Обсуждаются те же самые вопросы, что и на ежедневном скрам-совещании и дополнительно к ним:

1) Какой функционал удалось реализовать каждой команде с момента предыдущего ежедневного совещания?

2) Что каждая команда реализует к следующему ежедневному совещанию?

3) Имеются ли затруднения, которые мешают или замедляют работу каждой команды?

4) Требуется ли одной из команд выполнить решение части задач другой команды?

Совещание по обзору итогов спринта (Sprint review meeting) проводится после завершения спринта. К этому совещанию команда привлекает максимальное число зрителей и демонстрирует приращение функциональности разрабатываемого ПО всем заинтересованным лицам. Все члены команды из первой группы участвуют в совещании по итогам спринта. В рамках данного совещания, либо один человек демонстрирует результаты, достигнутые всей командой разработчиков или каждый из участников команды показывает, что было сделано за прошедший спринт. Причем не допускается показ незавершенной функциональности программного продукта (в рамках заданного приращения функциональности). Временные рамки этого совещания сверху ограничены четырьмя часами в зависимости от длительности итерации и приращения функциональности разрабатываемого программного комплекса.

Ретроспективное совещание (retrospective meeting) выполняется после завершения спринта. На данном совещании участники команды должны высказать свое мнение о прошедшем спринте и ответить на два ключевых вопроса:

1) Какие приемы позволяли осуществлять качественную работу в прошедшем спринте?
2) Какие улучшения требуются в следующем спринте?

Временные рамки данного совещания лежат в пределах от одного до трех часов, его основной целью является повышение качества процесса разработки, за счет фиксации удачных приемов, примененных в ходе выполнения данного спринта.

3.2. Методология TDD

Разработка через тестирование (test-driven development, TDD) представляет собой технологию разработки ПО, основанную на использовании весьма непродолжительных по времени итераций разработки. При использовании данной технологии вначале создается тест, при помощи которого можно проверить корректность вносимого в программу изменения, а затем лишь реализуется программный код, который должен успешно проходить тест и в завершении итерации производится рефакторинг вновь добавленного кода с целью приведения его к используемым при разработке проекта стандартам.

При своем создании (1999 г.) TDD рассматривалась как один из вариантов методологии экстремального программирования и была сильно интегрирована с концепцией «в начале тест», используемой в рамках данной методологии. Но в течение достаточно небольшого срока TDD сформировалась в виде отдельной методологии.

Функциональным назначением теста является проверка корректной работы написанного программного кода. Тестирование может быть произведено вручную, когда программист проверяет реакцию созданного кода на выполняемые действия. Однако такую проверку может выполнить и

компьютер, который будет эмулировать выполнение действий и производить оценку реакции на них со стороны разработанного ПО, т.е. тестирование в данном случае будет происходить в автоматическом режиме. При использовании данной методологии программирования корректность работы создаваемого ПО, прежде всего, будет определяться качеством используемых тестов и логикой их работы. В методологии TDD центральное место отводится созданию автоматических тестов и приобретению опыта в проведении процедуры тестирования. Написание тестов, на которых возлагается верификация созданного программного кода, крайне не желательно поручать человеку, разработавшему этот код. Поскольку в этом случае будет велика вероятность, что он создаст тест, который не обнаружит проблемных мест в разработанном им коде. Написанием тестов должен заниматься специальный участник команды разработчиков, называемый тестировщиком.

Использование методологии TDD предполагает, что один или несколько человек из участников команды разработчиков занимаются написанием автоматических тестов модулей, при помощи которых будут определятся критерии оценки правильности функционирования разработанного ПО, до создания самого программного кода. Тест должен включать в себя проверку определенных условий, которые либо не выполняются, либо выполняются. Если условия, проверяемые тестом, выполняются, то это означает его успешное прохождение. Из успешного проведения теста следует, что код, реализованный программистами, функционирует в соответствии с требованиями технического задания.

Достаточно часто при использовании методологии TDD разработка осуществляется с помощью имеющихся библиотек, содержащих шаблоны наиболее часто применяемых тестов. Использование таких готовых библиотек позволяет автоматизировать и ускорить процесс создания наборов тестов. Тесты модулей создаются для верификации функционирования наиболее критических частей разрабатываемого ПО. К таким частям, например, относится код, который в процессе разработки подвергается частой модификации или он влияет на работоспособность других элементов программного комплекса, или содержит значительное число зависимостей. При использовании методологии TDD необходимо, чтобы применяемая среда разработки обладала быстрой реакцией даже на достаточно небольшое изменение создаваемого кода.

Использование методологии TDD подразумевает не только проведение верификации разработанного ПО, но и оказывает существенное влияние на создание интерфейса разрабатываемого ПО. Созданные тесты помогают определить какую функциональность требуется реализовать в разрабатываемом ПО, чтобы обеспечить удобство конечного пользователя. Благодаря этому дизайн разрабатываемой системы создается значительно раньше ее полной программной реализации. При выполнении своей работы тестировщики, естественно, должны придерживаться тех же самых

стандартов, что и разработчики программного кода, реализующие проект. Графическое представление разработки ПО с использованием методологии TDD, описанное в [6], приведено на рис. 3.1.

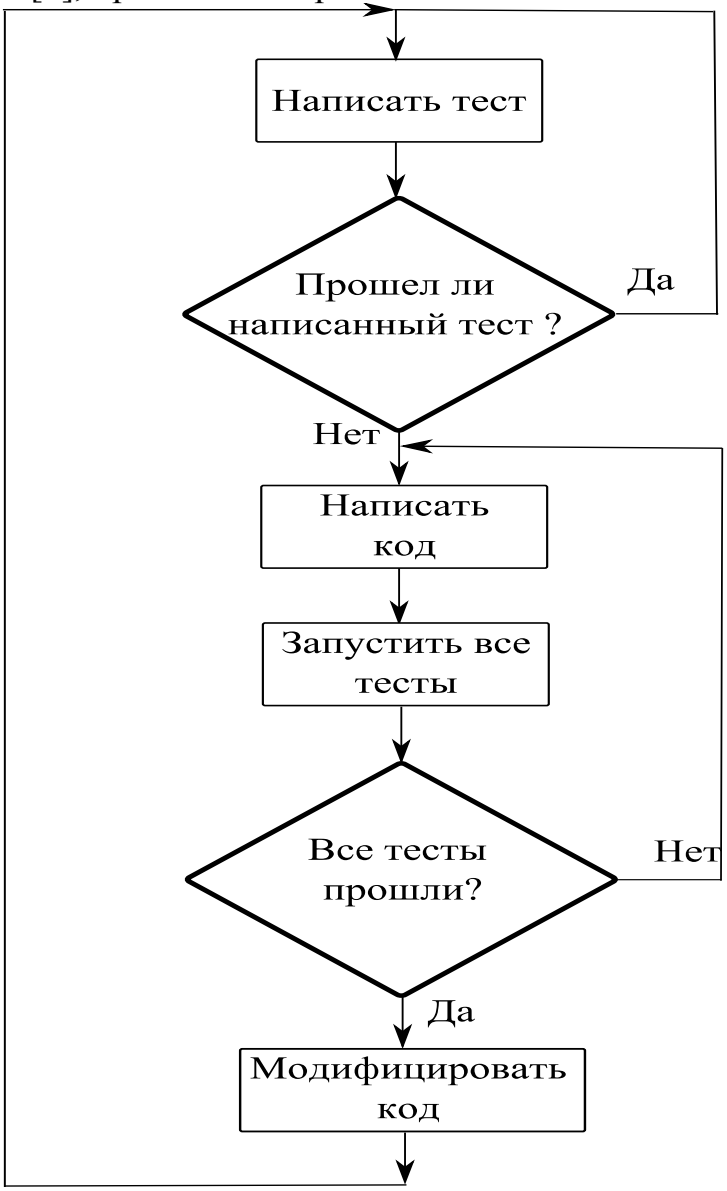

Рис. 3.1. Блок-схема разработки ПО с использованием методологии TDD

При использовании методологии TDD расширение функциональности (feature) уже существующего программного кода стартует с создания теста. Изначально, существующий программный код не должен пройти вновь созданный тест. Это вызвано тем, что на данном этапе программный код еще не подвергся модификации с целью расширения его функциональности, которая должна быть верифицируема уже созданным тестом. К успешному прохождению теста на данном этапе может привести одна из следующих двух причин:

1) Проверяемая тестом добавляемая функциональность уже реализована в программе.

2) Сам тест был реализован некорректно.

Следовательно, на данном этапе осуществляется проверка качества и самих разработанных тестов. Если тест все время показывает, что программный код реализован правильно, то он не приносит никакой пользы при разработке. Для качественного выполнения своей работы тестировщики должны иметь ясное представление о добавляемом в систему новом функционале. С этой целью анализируются вероятные схемы применения разрабатываемой системы, а также пожелания пользователей. Вновь сформулированные требования к разрабатываемой системе могут стать причиной замены имеющихся тестов.

Главной отличительной чертой методологии TDD от других способов разработки ПО является создание тестов, до написания самого программного кода. Эта особенность позволяет разработчику сфокусировать внимание на требованиях, предъявляемых к разрабатываемому ПО до самой реализации кода.

На этапе «запуска всех тестов» все вновь написанные тесты также изначально не должны проходить. Этот этап также служит для проверки качества самих тестов. Некорректность функционирования программного кода, которая выясняется при помощи тестирования, должна в этом случае иметь объясняемые причины. Это позволяет увеличить вероятность того, что тест в самом деле проверяет то, для чего он был создан.

На этапе написания кода в разрабатываемую систему добавляется новый функционал, причем выполнение теста должно будет подтвердить корректность произведенной модификации. Код, реализующий новый функционал, может быть и не оптимальным. Основной целью на данном этапе является прохождение теста, предназначенного для проверки кода, реализующего новый функционал программы. Писать изначально не очень оптимальный код при использовании методологии TDD допустимо, поскольку на последующих итерациях разработки будет происходить его оптимизация и улучшение. Добавляемый программный код должен допускать свою верификацию разработанным тестом, в нем не должна реализовываться функциональность, которую на данный момент можно проверить существующими тестами.

Вновь созданный программный код подвергается тестированию (всеми существующими для его проверки модульными тестами). Успешное прохождение тестов означает, что вновь написанный код соответствует требованиям технического задания.

Последним действием, выполняемым на одной итерации внешнего цикла процесса разработки, является рефакторинг созданного ПО. Под рефакторингом понимается процесс оптимизации разработанного ПО, который не оказывает влияние на организацию взаимодействия программы с пользователем.

При использовании методологии TDD процесс разработки ПО состоит из нескольких итераций, в каждой из которых выполняются вышеописанные действия. На каждой новой итерации производится добавление новой

функциональности разрабатываемого ПО к уже существующей. Рекомендуется объем изменений исходного кода ПО в рамках каждой итерации процесса разработки делать небольшим (выполнять модификации текста программы, которые обеспечивают добавление от 1 до 10 новых функций в разрабатываемую систему). Из этого правила имеется исключение - если при разработке ПО используются сторонние библиотеки, то небольшой объем изменений в исходном тексте программы, будет приводить к тестированию работы функций библиотеки (чего следует избегать, если имеется уверенность в корректной реализации этих функций).

Использование методологии TDD приводит к тому, что модульными тестами охватывается вся функциональность разрабатываемого программного комплекса. При использовании других методологий (когда вначале создается программный код, а лишь затем производится его тестирование) у разработчиков появляется возможность перейти к написанию кода, реализующего новую функциональность, не проверив корректность работы уже существующего программного кода.

Проверка того, что вновь созданный тест не подтверждает работоспособность существующего программного кода, показывает, что тест действительно корректно проверяет код. За этим тестирование следуют работы по добавлению кода в систему, реализующего новую функциональность. Данная последовательность разработки программного кода, при использовании методологии TDD, носит название «красный/зеленый/рефакторинг». Красным цветом здесь обозначается этап, когда существующий программный код не может пройти вновь созданные модульные тесты, а зеленым – когда он их успешно преодолевает.

Одно из направлений развития методологии TDD привело к появлению техники «разработка через приемочное тестирование» (Acceptance Test-driven development, ATDD). При использовании методологии ATDD тесты, которые далее используются при разработке ПО, создаются на основе требований, предъявляемых заказчиком к программному комплексу. Это позволяет гарантировать, что реализуемая система будет соответствовать требованиям, предъявляемым к ней заказчиком. При использовании методологии ATDD основной целью участников команды разработки является обеспечение прохождения приемочных (функциональных) тестов, что будет означать созданная система удовлетворяет требованиям заказчика.

Применение методологии TDD при написании программного кода уменьшает необходимость в использовании отладчика. Если верификация вновь добавленного кода не подтверждается тестами, то возврат к предшествующей версии работающего ПО и написание этого фрагмента системы заново, может быть более результативным, чем его отладка. Использование методологии TDD также оказывает существенное влияние и на стиль программирования: производится отказ от использования паттерна «одиночка» (singleton) и глобальных переменных, классы создаются более простыми для использования и не сильно связанными. Это объясняется тем,

что жестко связанный программный код, реализующий систему, требующую при своем запуске сложного процесса инициализации, гораздо сложнее поддается тестированию. Следование принципам TDD приводит к появлению в разрабатываемом коде небольших интерфейсов с ясной логической структурой, любой имеющий в системе класс в этом случае будет играть определенную, как правило, достаточно маленькую роль. Вследствие чего жесткость зависимостей между существующими в системе классами будет снижаться, а ее гибкость повышаться. Помимо тестирования в методологии TDD также используется контрактное программирование (design by contract), под которым понимается формирование требований к программному коду через утверждения (assertions).

Хотя при использовании методологии TDD приходится писать много исходного текста (сам код и тесты, его проверяющие), но затраты времени на реализацию проекта, как правило, оказываются меньше, чем при использовании других методологий. Необходимость поэтапного прохождения тестов, которые создаются раньше тестируемого ими программного кода, уменьшает вероятность обнаружения ошибок на последних этапах разработки, когда их устранения будет стоить весьма дорого.

Наличие тестов позволяет минимизировать вероятность появления дополнительных ошибок при выполнении рефакторинга кода. Это придает разработчикам проекта уверенность, что модификация существующего кода при добавлении архитектурных решений не приведет к нарушению работы уже нормально функционирующих частей. Следование принципам TDD приводит к написанию хорошо масштабируемого и гибкого кода, состоящего из большого количества модулей. Это происходит из-за того, что при использовании методологии TDD реализуемая система представляется разработчику в виде большого количества модулей относительно небольшого объема, которые создаются и тестируются отдельно друг от друга и лишь после этого происходит их объединение. Используемые в системе классы получаются более специализированными и содержащими меньшее количество элементов, они получаются достаточно гибко связаны между собой и реализуют интерфейсы, обладающие ясной логической структурой. Так как при использовании методологии TDD программный код создается для прохождения существующих тестов, то они полностью позволяют проверить все части программы и обнаружить незапланированное поведение разрабатываемой системы.

При использовании методологии TDD автоматизированные тесты могут выступать в качестве своеобразной документации. Используемая при разработке проекта документация, а также комментарии к коду могут устареть. Это может вызвать затруднения разработчиков при последующих модификациях проекта. В той же самой ситуации по результатам выполнения теста, в отличии от документации, можно будет понять, что он устарел.

Поскольку при написании кода большинства крупных программных проектов применяется объектно-ориентированная парадигма программиро-

вания, то при реализации системы с использование разных технологий разработки (в том числе и TDD) приходится решать проблему видимости кода. С одной стороны для существующего набора тестов необходимо обеспечить полный доступ ко всему программному коду. С другой стороны, такой краеугольный принцип объектно-ориентированного программирования, как инкапсуляция нарушаться не должен. По этой причине юнит-тесты, как правило, располагаются в том же модуле, что и сам проверяемый код.

При создании модульных тестов может потребоваться проведение дополнительной работы, поскольку из них не всегда можно обеспечить доступ к закрытым методам и полям классов. Способ решения этой проблемы будет зависеть от особенностей конкретного языка программирования, используемого при разработке. При применении, например, языка Java для обеспечения доступа к закрытым полям класса можно использовать отражение (reflection). В .NET Framework для решения этой проблемы могут использоваться частичные классы (partial classes). Также юнит-тесты можно встроить во вложенные классы, с целью обеспечения их доступа к элементам внешних и внутренних классов.

Из конечного варианта разрабатываемого программного продукта необходимо удалить участки кода, отвечающие исключительно за его тестирование. Для этих целей обычно используют директивы условной компиляции, поддержка которых реализована практически во всех языках программирования. Использование директив условной компиляции означает, что конечный вариант созданного программного кода будет не полностью совпадать с вариантом, который прошел верификацию модульными тестами. В этом случае на конечной версии сборки разработанной программы переходят к систематическому запуску интеграционных тестов, с целью обнаружения фрагментов кода в своем функционировании неявно опирающихся на существовавшие модульные тесты.

Среди разработчиков, использующих методологию TDD, нет единства в том, следует ли подвергать тестирование защищенные поля и методы классов. Одна часть разработчиков придерживается мнения, что следует конструировать тесты так, чтобы они могли обеспечить проверку правильности функционирования класса только через открытые элементы (методы, свойства и т.д.), которые составляют интерфейс класса, так как детали внутренней реализации класса могут быть изменены и это не должно отразиться на используемых тестах. Другая часть считает, что часть функциональности создаваемой системы может быть реализована и в закрытых методах и их неявное тестирование через открытый интерфейс класса только приведет к усложнению процесса разработки, поскольку использование принципов TDD предполагает, что верификации должны подвергаться модули минимально возможного размера.

Отдельный модуль может содержать любое число юнит-тестов, предназначенных для проверки корректности его функционирования. Не следует допускать, чтобы код юнит-тестов устанавливал сетевые соединения

или выходил за границы процесса, так как это приведет к значительным затратам времени на их прохождение и участники команды реже будут запускать наборы тестов. Появление зависимостей от элементов разрабатываемой системы, расположенных во внешних модулях изменяет также и тип используемых тестов, превращая их из модульных в интеграционные. Если при этом поведение хотя бы одного модуля в цепочке будет отличаться от запланированного, то могут возникнуть проблемы с локализацией месторасположения кода, реализующего неправильную функциональность.

В случае использования в разрабатываемой системе обращения к внешним ресурсам (веб-сервисам, базам данных и т.д.), обычно из нее выделяют подлежащую тестированию часть. Происходит это в два этапа:

1) Объявляется интерфейс, основной задачей которого является обеспечение доступа к внешним процессам.

2) Осуществляются две реализации объявленного интерфейса. Первая реализация интерфейса должна обеспечить доступ к используемому внешнему ресурсу, а вторая представлять из себя mock- или fake- объект.

Fake-объекты тестируют корректность поведения разработанного кода путем записи в лог сообщений вида «Объект *имя_объекта* сохранен». В mock-объектах содержатся утверждения (assertion), при помощи которых осуществляется оценка корректности функционирования разработанного ПО.

Одним из приемов тестировании является настройка методов fake- и mock-объектов таким образом, чтобы они всегда возвращали одни и те же одинаковые величины. Также fake- и mock-объекты могут произвести эмуляцию возникновения ошибки с целью детального анализа функционирования кода, производящего обработку ошибок.

Применение mock- и fake-объектов для эмуляции внешнего окружения приведет к тому, что реальный программный код, осуществляющий взаимодействие с внешним источником не будет подвергнут верификации юнит-тестами. Для того чтобы не допустить возникновения ошибок в этом коде, необходимо провести его верификацию с помощью интеграционных тестов, которые должны отделяться от существующих юнит-тестов. Поскольку для проверки правильности функционирования кода требуется гораздо меньшее количество интеграционных тестов по сравнению с юнит-тестами, то их можно запускать гораздо реже.

Если при выполнение интеграционного теста произошло изменение в записях БД, то вне зависимости от исхода прохождения теста (успешного или не успешного) необходимо произвести откат к первоначальному состоянию (до запуска теста) БД. Это действие может быть реализовано следующими способами:

1) Присутствующим практически во всех библиотеках для тестирования методом `TearDown.`

2) Конструкцией для обработки исключительных ситуаций (`try... catch... finally` в языках C++, C#; `try... except` в языке Object Pascal и т.д.)

3) Транзакции баз данных.

4) Откат после проведения интеграционных тестов к созданному перед их запуском снимку (snapshot) БД.

Можно выделить следующие недостатки использования TDD:

1) К использованию TDD достаточно сложно привыкнуть.

2) Существуют задачи, которые не могут быть решены при использовании данной методологии. Например, если требуется обеспечить безопасность данных, то помимо корректного поведения кода (что можно протестировать), необходимо также обеспечить выполнение определенных процедур человеком.

3) TDD затруднительно применять в тех случаях, когда для верификации разработанного ПО используются функциональные тесты (разработка пользовательских интерфейсов, доступ к БД, сетевое взаимодействие с учетом особенностей конфигурации сети и т.д.). Основное внимание при использовании TDD уделяется тестированию отдельных модулей, а внешнее окружение в тестах представляется при помощи mock-fake- объектов.

4) При отсутствии опыта использования TDD на реализацию разрабатываемой системы потребуется больше времени.

5) При совмещение ролей разработчика и тестировщика в одном лице, в случае неправильной им трактовки положений технического задания и проверяемый модуль и тест для его проверки будут содержать ошибки.

6) Появление дополнительных накладных расходов на написание юнит-тестов.

7) Риск ослабления контроля за качеством процесса разработки, так как значительное количество юнит-тестов может придать уверенность, что процесс написания ПО протекает так как надо.

3.3. Методология MSF

Microsoft Solutions Framework (MSF) представляет из себя методологию программирования первоначальный вариант которой, открытый и для сторонних разработчиков, был представлен фирмой Microsoft в 1994 г. Методология MSF была создана на основе обобщения опыта, приобретенного корпорацией Microsoft, в процессе создания ПО и описывает организацию рабочих процессов необходимую для успешного завершения работы над проектом. Методологию MSF можно представить в виде взаимосвязанного набора правил, концепций и моделей. К MSF логически примыкает и дополняет ее Microsoft Operations Framework (MOF). На основе базовых методов MSF в корпорации Microsoft были разработаны методики для специализированного и прикладного использования. Для подтверждения умения использования методологии MSF при разработке ПО можно сдать

один из сертификационных экзаменов Microsoft из данной предметной области, например, MCTS 74-131. Имеется несколько прикладных вариантов MSF, причем наиболее популярными являются два из них:

1) Методика внедрения решений в области управления проектами разработки ПО.

2) Методика управления проектами разработки ПО на основе методологий MSF и Agile.

В чистом виде методология MSF не используется для разработки ПО даже в самой корпорации Microsoft, поэтому практическую ценность приобретают различные прикладные варианты этой методологии. Например, в проектах Microsoft Consulting Services, применяется комбинация методологий Agile и MSF. Различные прикладные варианты методологии MSF чисто внешне обладают весьма существенными различиями, однако, в их основе всегда лежит «чистый вариант» методологии MSF, который и определяет подход к процессу разработки ПО.

MOF по сути представляет техническое руководство по созданию ПО с использованием технологий и продуктов Microsoft. Следования требованиям, изложенным в MOF, позволяет обеспечить необходимый уровень надежности (reliability), доступности (availability), удобства сопровождения (supportability) и управляемости (manageability) разрабатываемого ПО. В MOF освещаются вопросы, которые связаны с управлением процессами и персоналом, а также с технологиями, используемыми при разработке ПО в распределенных (distributed), разнородных (heterogeneous) и сложных (complex) IT-средах. Основу MOF составляют методики, содержащиеся в Information Technology Infrastructure Library (ITIL), которая была создана при участии агентства правительства Великобритании по компьютерным и информационным технологиям (Central Computer and Telecommunications Agency).

Залогом успешного завершения проекта по созданию ПО, в условиях отводимого на него фиксированного бюджета и времени, является применение испытанной методологии разработки. Испытанные методики планирования, проектирования, создания и внедрения качественных IT-решений предлагаются в рамках методологии MSF. Для того чтобы при создании ПО методологию MSF могли использовать организации и группы разных размеров, она не содержит жестких инструкций и обладает большой гибкостью и масштабируемостью. Методологию MSF можно представить в виде набора принципов, дисциплин и моделей, которые описывают процесс разработки ПО, включая такие аспекты, как руководство персоналом, управление технологическими элементами и процессами. Предметная область MSF включает в себя три дисциплины (управление проектами; управление рисками; управление подготовкой), а также содержит описание двух моделей (модель процессов; модель проектной группы).

Способы организации команды участников, позволяющую повысить вероятность успешного окончания работы над проектом, определяет модель проектной группы MSF (MSF Team Model). В рамках данной модели задаются

ролевые кластеры, определяются, какие задачи они должны решать в ходе работы над проектом, а также даются рекомендации участникам команды разработчиков, необходимые для успешного завершения работы над проектом. Недостатки в организации традиционной иерархической модели команды участников работы над проектом послужили причиной создания MSF Team Model разработка которой продолжалась в течение нескольких лет. Согласно MSF Team Model группы, принимающие участие в разработке ПО, должны представлять небольшие многопрофильные команды, члены которых дополняют области компетенций друг друга и несут коллективную ответственность за порученный фронт работы. Это позволяет более четко сконцентрировать внимание на требованиях проекта. Всю проектную группу, состоящую из нескольких команд разработчиков, должен объединять единый взгляд на проект, а также стремление к его успешной реализации, высокие требования, предъявляемые к качеству работы, и, как следствие, желание самосовершенствоваться.

Согласно методологии MSF для успешной работы команды необходимо следовать следующим принципам:

1) Распределять ответственность при фиксации отчетности о проделанной работе по проекту.

2) Наделять участников команды разработчиков необходимыми полномочиями.

3) Концентрировать усилия на бизнес-приоритетах.

4) Достигать единого взгляда на проект со стороны всех участников процесса разработки, что предполагает с их стороны понимание задач и целей написания данного ПО.

5) Гибкость, предполагает готовность к изменению или уточнению требований к разрабатываемой системе в ходе работ по реализации проекта, что позволяет не допустить потери контроля за процессом разработки и снижения качества выпускаемого ПО.

6) Поощрение свободного обмена информацией как внутри команды разработчиков, так и с другими лицами, заинтересованными в успешном окончании проекта.

Ключевые концепции (key concepts) методологии MSF обеспечивают успешное применение MSF Team Model. В рамках методологии MSF считается, что одним из условий успешного завершения работы над проектом, является следование следующим шести ключевым концепциям:

1) Участники работы над проектом должны представлять из себя команду соратников.

2) Группа разработчиков должна сконцентрировать усилия на выполнении требований к разрабатываемой системе, предъявляемых заказчиком.

3) Участники работы над проектом должны быть нацелены на конечный результат.

4) Установка на отсутствие дефектов в разрабатываемой системе.

5) Участники работы над проектом должны стремиться к самосовершенствованию (освоению новых технологий программирования и приобретению соответствующих практических навыков).

6) Для повышения эффективности работы в целом необходимо, чтобы все команды, участвующие в процессе создания ПО, были заинтересованы в успешном окончании каждого из этапов процесса разработки.

Помимо модели проектной группы и ключевых концепций методология MSF также базируется на постулате о шести качественных целях. Необходимость достижения этих целей обуславливает модель проектной группы, а также с большой степенью вероятности гарантирует успешное завершение работ по реализации проекта (в рамках отведенного на него бюджета и времени). В методологии MSF ответственность за успешную реализацию проекта возлагается на всю команду, но любой из ее ролевых кластеров работает над достижением одной из поставленных шести целей. Модель проектной группы MSF составляют следующие ролевые кластеры [8]:

1) управление программой;
2) управление продуктом;
3) разработка;
4) тестирование;
5) управление релизом;
6) удовлетворение потребителя.

Ролевые кластеры, иногда называемые просто ролями, отвечают за отдельные области компетенции (functional areas) и должны обеспечивать достижение целей и решение задач, связанных с этими областями. Использование методологии MSF позволяет процесс разработки ПО сделать масштабируемым.

Ролевой кластер «управление программой» (program manager) отвечает за проектирование архитектуры приложения и использование административных служб. Кластер «управление продуктом» (product manager) обеспечивает представление интересов заказчика, определяет бизнес-приоритеты и проводимую маркетинговую политику. Кластер «разработка» (developer) обеспечивает создание инфраструктуры проекта, непосредственную работу по реализации разрабатываемой системы и технологические консультации. Областью ответственности кластера «тестирование» является планирование и создание тестов, а также анализ по результатам тестов качества разработанного ПО. Ролевой кластер «управление выпуском» (release manager) отвечает за создание инфраструктуры, обеспечивающий бизнес-процессы по выпуску и сопровождению разработанной системы. Ролевой кластер «удовлетворение пользователя» (user experience) отвечает за интерфейс разработанной системы, ее эргономику, обучение работы с ней конечных пользователей и осуществление их технической поддержки.

Так как в зависимости от профессиональных компетенций участников команды, масштаба реализуемого проекта и его сложности один человек может выполнять несколько ролей или, наоборот, один ролевой кластер может

включать себя несколько участников, то количество членов в команде разработчиков не обязательно должно быть кратно шести.

Минимальная численность команды разработчиков согласно методологии MSF может достигать всего трех человек. Модель проектной группы MSF не подразумевает, что на каждый ролевой кластер должно приходится более одного сотрудника. Основная идея заключается в том, что участники команды, в процессе работы над проектом, должны добиться поставленных шести целей, поэтому возможно совмещение отдельных ролей некоторыми из них. Однако, если на каждый ролевой кластер, выделяется как минимум один человек, то это позволяет более качественно исполнить эту роль. Но при разработке достаточно небольших проектов такое выделение часто бывает экономически нецелесообразно. Поэтому в небольших проектных группах обычно выполняется совмещение ролей, при этом необходимо выполнение следующих условий:

1) Роль команды разработчиков не должна быть совмещена с любой другой ролью.

2) Следует избегать совмещения ролей, в которых потенциально заложены определенные конфликты интересов.

На практике, с целью уменьшения затрат на реализацию проекта, часто выполняют совмещение ролей, при этом успех такого объединения в значительной степени будет определяться опытом и профессиональными компетенциями участников команды разработчиков. При рациональном варианте объединения ролей разработки одним человеком и контролем за возникающими вследствие такого объединения рисками можно в значительной степени минимизировать возможные негативные последствия данного шага.

В рамках методологии MSF не даются точные указания по управлению процессом разработки проекта и не объясняются разнообразные приемы, использующиеся участниками команды, а просто формируется подход к управлению процессом разработки проектов, отличительными чертами которого являются:

1) Распределенная между лидерами ролевых кластеров внутри одной команды ответственность за управление процессом разработки проекта. Любой участник команды несет ответственность за успешное окончание работ по реализации проекта и за качество создаваемого ПО.

2) Профессиональные менеджеры не осуществляют функции контроля за работой команды, а выступают в качестве наставников и консультантов участников команды. Это положение согласно методологии MSF должно повысить эффективность работы команды в целом. Таким образом, должность менеджера проекта, как таковая, в случае применения для разработки методологии MSF будет отсутствовать.

Согласно MSF Model Team команды разработчиков с большим числом участников (10 человек и более) должны делиться на небольшие многопрофильные группы направлений (feature teams). Эти группы выполняют работу

параллельно, периодически производя синхронизацию друг с другом. Если разработка проекта требует выделения большого количества ресурсов, то формируют функциональные группы (functional teams), на основе которых в дальнейшем создаются ролевые кластеры. Применение ролевых кластеров не означает, что организации, использующие при разработке ПО методологию MSF, должны иметь специальную структуру или какие-то обязательные должности в штате. В различных группах разработчиков и организациях состав ролей может варьироваться в широком диапазоне. Обычно роли распределяются между отдельными подразделениями одной организации, однако иное определенное количество ролей отводится консультантам и партнерам, работающим вне организации или даже сообществу потребителей. Одним из факторов успешного применения методологии MSF для разработки ПО является четкое закрепление работников за определенным ролевым кластером, определение их должностных обязанностей и частей системы, которые они должны разработать.

Использование MSF Model Team, определяющей структуру команды разработчиков и обеспечивающей эффективность ее функционирования, является существенным фактором, который может оказать влияние на успешное завершение работ по реализации проекта. Однако на успешное завершение работ по реализации проекта оказывают влияние также и другие факторы помимо использования MSF Model Team.

Модель процессов MSF (MSF process model) определяет общую методологию по разработке и внедрению ПО. Отличительной чертой MSF process model является гибкость и отсутствие жестко навязанных процедур разработки, что позволяет использовать данную модель при реализации весьма широкого диапазона ИТ-проектов. MSF process model представляет из себя комбинацию водопадной и спиральной модели разработки (рис. 3.2).

В третьей и более поздних версиях модель процессов MSF описывает весь жизненный цикл ПО, начиная с момента начала работы над проектом и заканчивая внедрением и сопровождением разработанной системы. Благодаря этому команды разработчиков могут сконцентрировать усилия на бизнес-отдаче (business value) от реализуемой системы, поскольку она наступает лишь после завершения работ по внедрению и начала эксплуатации разработанного ПО.

MSF process model ориентируется на так называемые «вехи» (milestones), представляющие из себя ключевые точки проекта, в которых дается оценка, достигнуты ли требуемые результаты (промежуточные или конечные) в процессе разработки. Оценка и анализ достигнутых результатов может быть выполнен в форме ответов на следующие вопросы: «Достигнуто ли в команде разработчиков единое видение целей и задач проекта?», «Выработан ли план реализации проекта?», «Отвечают ли характеристики разработанной системы требованиям технического задания?», «Доволен ли заказчик функционированием разработанной системы» и т.д.?

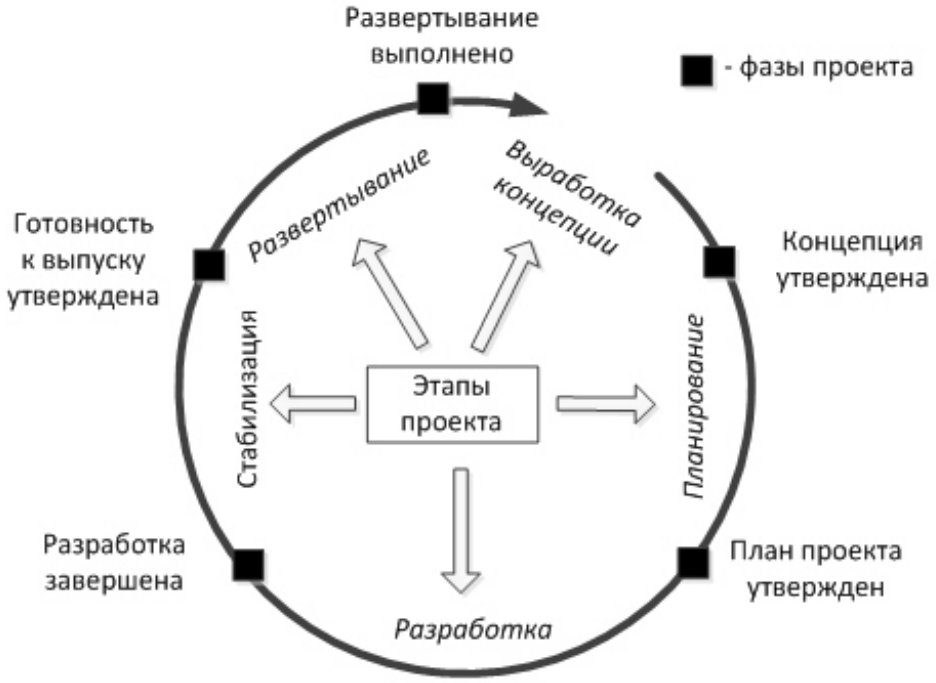

Рис. 3.2. Разработка ПО с использованием методологии MSF

MSF process model позволяет произвести учет изменений и уточнений требований к разрабатываемой системе. Данная модель исходит из того, что процесс разработки ПО должен состоять из небольших итераций, в которых производится наращивание функциональности разрабатываемой системы, начиная от ее простейших версий и заканчивая вариантом, который полностью удовлетворяет всем требованиям технического задания (с учетом внесенных в него дополнений и изменений в процессе разработки).

MSF process model интегрируется с основными принципами MSF и имеет следующие отличительные черты:

1) Вехи и фазы представляют основу процесса разработки.

2) К процессу разработки ПО применяется итеративный подход.

3) Производится интеграция подходов к разработке и внедрению создаваемого ПО.

Модель процессов MSF определяет следующие основные фазы процесса разработки [7]:

1) Выработка концепции (Envisioning).

2) Планирование (Planning).

3) Разработка (Developing).

4) Стабилизация (Stabilizing).

5) Внедрение (Deploying).

Также имеется значительное число промежуточных вех, которые используются для разбиения больших этапов процесса разработки на более мелкие, а также служат в качестве индикаторов прогресса работ по реализации ПО.

Для любой фазы MSF process model определяется:

1) Что должно являться конечным результатом выполнения этой фазы проекта (какие артефакты разрабатываемой системы должны быть созданы на этой фазе).

2) Определяется над какими частями создаваемой системы должны работать ролевые кластеры.

В рамках методологии MSF рекомендуется начинать работы по реализации проекта с написания основной функциональности системы, ее тестирования и внедрения. Причем разработка архитектуры приложения и его интерфейса, определение и уточнение плана работ по реализации системы, написание программного кода и документации к нему выполняются, как правило, в итеративных циклах. Далее производится наращивание функциональных возможностей разрабатываемой системы по сравнению с первоначальным вариантом. Данная стратегия в рамках методологии MSF называется стратегией версионирования. Несмотря на то, что для небольших по масштабу проектов можно обойтись созданием только одной версии разрабатываемого ПО, рекомендуется все равно создавать несколько таких версий.

Поскольку на каждой итерации происходит модификация программного кода с целью наращивания функциональности разрабатываемой системы и учета возможных изменений и дополнений требований, первоначально сформулированных в техническом задании, то требуется достаточно гибкий способ ведения документации. «Живые» документы (living documents), точно так же, как и программный код и требования, предъявляемые к разрабатываемой системе, должны модифицироваться по мере выполнения работ по реализации проекта. В рамках методологии MSF имеется несколько шаблонов стандартных документов, которые являются артефактами определенной стадии разработки ПО и могут быть полезны при планировании работ по реализации проекта, а также осуществления контроля за ходом выполнения этих работ.

Разработанное ПО не представляет из себя никакой бизнес-ценности, пока оно не будет внедрено. Поэтому MSF process model описывает весь жизненный цикл разработки ПО, включая этап внедрения, после которого созданное ПО приобретает бизнес-ценность и начинает окупать средства, затраченные на его разработку.

Третьим существенным фактором, предопределяющим успешное окончание работ по реализации проекта, согласно методологии MSF является дисциплина управления рисками (risk management). Согласно методологии MSF возникающие в ходе выполнения работ по реализации проекта изменения и вызванные этим неопределенности представляют неотъемлемую часть жизненного цикла ПО. Дисциплина управления рисками MSF (MSF risk management discipline) предполагает, что в условиях возникающей неопределенности в ходе работы над проектом необходим превентивный подход к работе с появляющимися рисками и их непрерывная оценка. Основываясь на

оценке рисков можно вносить адекватные изменения в план выполнения работ по реализации проекта на всем протяжении жизненного цикла ПО. Дисциплина управления рисками MSF предоставляет идеи, принципы и рекомендации, которые дополняются детализированным описанием процесса, позволяющим успешно осуществить активное управление рисками. В рамках данного процесса производится определение и анализ потенциальных рисков, планирование мер по их профилактике и минимизации негативных последствий в случае их наступления, отслеживания текущего состояния рисков, а также анализа достигнутых результатов.

В рамках методологии MSF фактически дается описание управления проектом. При этом под проектом в MSF понимается ограниченная определенным временным промежутком деятельность, целью которой является создание уникального ПО. В свою очередь под управлением проектом в MSF понимается совокупность знаний, практических навыков и приемов, инструментальных средств, которые используются для реализации целей проекта, с учетом лимитированных бюджета и срока, а также требований, предъявляемых к качеству разрабатываемой системы. Выделяемые в рамках методологии MSF области ответственности, навыков и деятельности по управлению проектами приведены в табл. 3.3.

Можно дать следующие характеристики управления проектами в MSF:

1) Основная часть ответственности по управлению проектом поручается ролевому кластеру «Управление программой».

2) Масштабированная модель проектной команды MSF применяется в больших проектах, когда требуется, чтобы работы по управлению проектом осуществлялись на нескольких уровнях.

3) Иногда, для больших и сложных проектов, возникает необходимость в привлечении отдельного специалиста или даже группы по управлению проектом.

Взаимосвязь между ресурсами проекта (финансовыми и людскими), временем, отводимым на разработку и внедрение ПО, а также требованиями, предъявляемыми к разрабатываемой системе, можно представить в виде так называемого «треугольника компромиссов» (рис. 3.3). Одной из задач управления проектом согласно методологии MSF является нахождение оптимального баланса между ресурсами и временем затраченными на разработку, и созданным функционалом реализованной системы.

После достижения баланса в треугольнике компромиссов модификация на любой из его сторон для сохранения равновесия потребует изменения на одной или двух других сторонах и/или на изначально измененной стороне.

Таблица 3.3

Области ответственности в методологии MSF

Область управления проектами	Описание
Планирование и мониторинг проекта, контроль за изменениями в проекте (Project planning / Tracking / Change Control)	Интеграция и синхронизация планов проекта; организация процедур и систем управления и мониторинга проектных изменений
Управление рамками проекта (Scope Management)	Определение и распределение объема работы (рамок проекта); управление компромиссными решениями в проекте
Управление календарным графиком проекта (Schedule Management)	Составление календарного графика исходя из оценок трудозатрат, упорядочивание задач, соотнесение доступных ресурсов с задачами, применение статистических методов, поддержка календарного графика
Управление стоимостью (Cost Management)	Оценки стоимости исходя из оценок временных затрат; отчетность о ходе проекта и его анализ; анализ затратных рисков; функционально-стоимостной анализ (value analysis)
Управление персоналом (Staff Resource Management)	Планирование ресурсов; формирование проектной команды; разрешение конфликтов; планирование и управление подготовкой
Управление коммуникацией (Communications Management)	Коммуникационное планирование (между проектной группой, заказчиком/спонсором, потребителями/пользователями, др. заинтересованными лицами); отчетность о ходе проекта
Управление рисками (Risk Management)	Организация процесса управления рисками в команде и содействие ему; обеспечение документооборота управления рисками
Управление снабжением (Procurement)	Анализ цен поставщиков услуг и/или аппаратного/программного обеспечения; подготовка документов об инициировании предложений (requests for proposals - RFPs), выбор поставщиков и субподрядчиков; составление контрактов и переговоры об их условиях; договора; заказы на поставку и платежные требования
Управление качеством (Quality Management)	Планирование качества, определение применяемых стандартов, документирование критериев качества и процессов его измерения

В реальных проектах может производится фиксация одного из ресурсов, тогда в ходе создания системы изменению будут подлежать только два

оставшихся ресурса. Данная ситуация фиксируется матрицей компромиссов, внешний вид которой приведен на рис. 3.4.

Рис. 3.3. «Треугольник компромиссов»

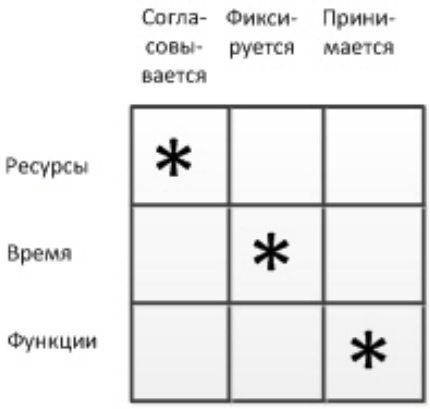

Рис. 3.4. Матрица компромиссов

Например, если фиксируются ресурсы, то возможно внесение корректив в функциональные возможности, реализуемые разрабатываемой системой, и согласование календарного плана выполнения проекта.

В MSF описывается еще одна ключевая дисциплина, которая называется «управление подготовкой». Данная дисциплина пытается описать процесс управления теоретическими знаниями, профессиональными навыками и умениями, которые требуются участникам команды разработчиков для успешного планирования, реализации и внедрения создаваемого ПО. Дисциплина управления подготовкой опирается на фундаментальные принципы MSF и предлагает рекомендации по использованию превентивного подхода к процессу управления знаниями разработчиков на протяжении всего жизненного цикла ПО. Также в данную дисциплину входит и планирование процесса управления подготовкой. Дисциплина управления подготовкой, подкрепляемая проверенными практическими методиками, предоставляет теоретическую базу отдельным специалистам и даже целым проектным группам для осуществления данного процесса.

4. ТЕХНОЛОГИЯ РАЗРАБОТКИ ПО С ИСПОЛЬЗОВАНИЕМ ИНСТРУМЕНТАЛЬНЫХ СРЕДСТВ MICROSOFT

4.1. Сервер Team Foundation *Server (TFS)*

4.1.1. Общие сведения о TFS

Microsoft Visual Studio Team Foundation Server (TFS) используется для организации совместной командной работы по разработке ПО. При помощи TFS можно выполнять:

1) управление проектами;

2) производить наблюдение за текущим состоянием рабочих элементов проекта;

3) осуществлять контроль версий разрабатываемого ПО;

4) автоматизировать процесс тестирования разрабатываемого ПО;

5) выполнять автоматического построении сборок ПО;

6) автоматически генерировать отчетность.

С точки зрения архитектуры TFS 2012 представляет трехуровневое сервис-ориентированное ПО (рис. 4.1). СУБД MS SQL Server 2012 предназначена для управления уровнем данных. Уровень приложения реализуется с помощью технологии ASP.NET и размещается в среде IIS (Internet Information Services), представляющей из себя набор серверов для нескольких служб Интернета от компании Microsoft.

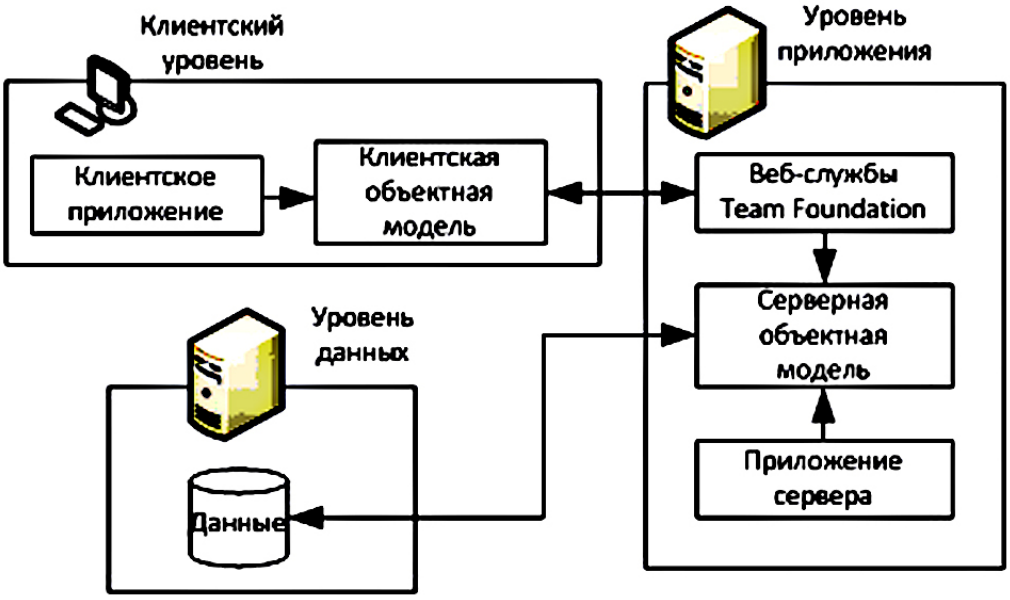

Рис. 4.1. Архитектура Team Foundation Server

С логической точки зрения TFS является веб-приложением, которое состоит из нескольких веб-сервисов (веб-служб), функционирующих на уровне приложения и обеспечивающих взаимодействии с клиентским уровнем. Эти сервисы обеспечивают функциональность TFS. На уровне приложения функционируют следующие веб-сервисы: построения сборок,

управления версиями, отслеживания рабочих элементов проекта, лаборатория тестирования Lab Management и службы платформы TFS. Интерфейсом прикладного программирования (API - application programming interface) для TFS выступает серверная объектная модель, которую обычно используют, когда возникает потребность в расширении функциональности данного сервера.

На уровне данных располагаются несколько реляционных БД и хранилищ данных (конфигурации сервера, аналитики, коллекции командных проектов и непосредственно хранилища данных). Сведения о командных проектах помещаются в реляционные БД и через заданные промежутки времени переписываются в хранилище данных.

Для работы на клиентском уровне можно применять либо веб-браузер, либо IDE Microsoft Visual Studio. Причем желательно, чтобы версия Visual Studio соответствовала версии TFS, например, с TFS 2012 применять VS 2012. В этом случае не будет наблюдаться никаких проблем при интеграции этих приложений. Для взаимодействия клиентского уровня с уровнем приложений опять используют серверную объектную модель и применяют веб-сервисы. Также на клиентском уровне поддерживается интеграция с приложениями Microsoft Office.

4.1.2. Развертывание TFS

Существуют следующие варианты установки TFS:

1) на одном сервере;
2) на нескольких серверах;
3) в одном домене или рабочей группе;
4) в нескольких доменах.

В самом простом случае компоненты, представляющие логический уровень TFS, располагаются на одном физическом сервере (рис. 4.2). К этим компонентам относятся: Team Foundation Server, SQL Server, Reporting Services и Widows SharePoint Services. Общее число пользователей системы в конфигурации с одним сервером, как правило, не превышает 50 человек. При такой конфигурации построение (Team Foundation Build) и тестирование приложений может быть произведено как на клиентских машинах, так и на сервере.

При использовании простой серверной топологии компоненты представляющие логический уровень TFS, опять размещаются на одном физическом сервере (рис. 4.3). Однако такая конфигурация позволяет более лучше учесть и распределить дополнительную нагрузку на процессорные мощности, создаваемую ПО, предназначенным для построения и тестирования приложений.

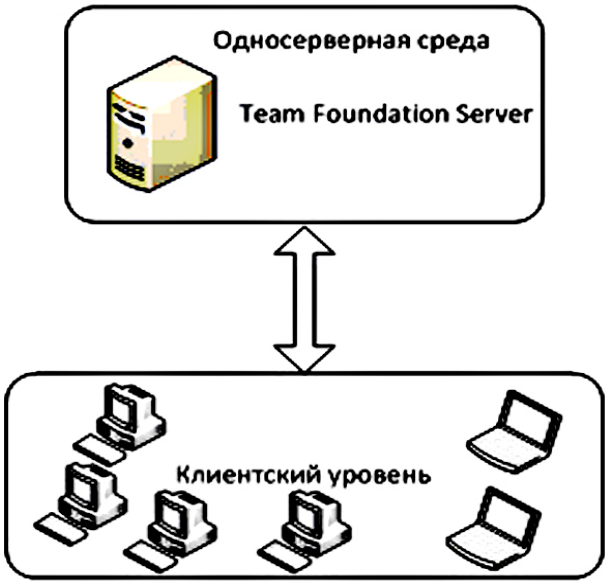

Рис. 4.2. Простейшая серверная топология TFS

На рис. 4.3 веб-сервисы и БД для TFS располагаются на одном физическом сервере, но для сервисов построения и тестирования выделяется отдельный компьютер. В такой конфигурации доступ к TFS может быть осуществлен только с компьютеров, принадлежащих тому же домену или рабочей группе, что и компьютер, на котором установлен TFS. Общее число пользователей для данной конфигурации обычно не превышает 100.

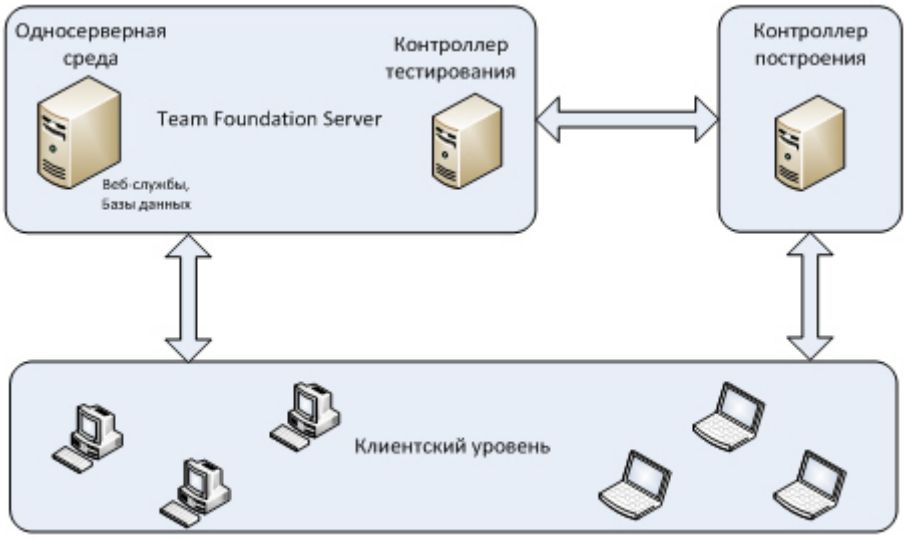

Рис. 4.3. Простая серверная топология TFS

В случае, если логические компоненты уровня данных и приложений размещаются на нескольких серверах (двух или более), то имеет место переход к топологии средней сложности (рис. 4.4). На рис. 4.4 сервисы уровня приложений для TFS устанавливаются на одном сервере, а БД для TFS помещаются на специально выделенный сервер.

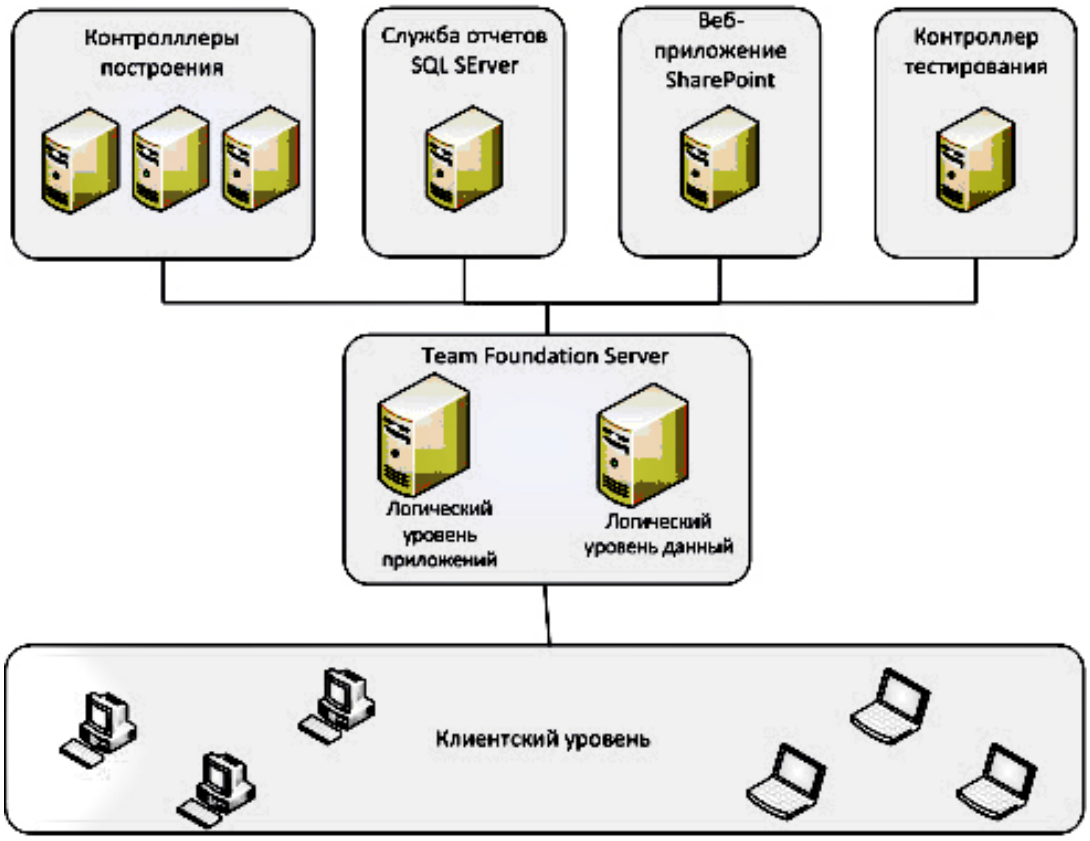

Рис. 4.4. Серверная топология TFS средней сложности

На специально выделенных серверах также располагаются веб-приложение SharePoint и экземпляр служб отчетов СУБД SQL Server. Для любого командного проекта, в данной конфигурации, создается портал, который размещается в приложении SharePoint. Также на специально выделенных серверах располагаются сервис Team Foundation Build и тестовые контроллеры. Обычно число пользователей в данной конфигурации не превышает 1000.

Для осуществления управления командной разработкой в Team Foundation Server размещается одна или несколько коллекций командных проектов, каждая из которых может состоять из нуля или более проектов.

4.1.3. Шаблоны командных проектов в TFS

Командный проект включает в себя коллекцию кода, тестов и построений, рабочих элементов. Элементы этой коллекции охватывают все артефакты, которые используются в жизненном цикле ПО. Командный проект создается на базе шаблона, представляющего из себя набор XML-файлов, в которых производится детализация процесса построения проекта. В TFS 2012 доступны следующие шаблоны командных проектов (рис. 4.5):

1) MSF for CMMI Process Improvement 6.2, предназначенный для команд с большим числом участников и строго формальным подходом к процессу управления проектами на базе модели CMM/CMMI.

2) MSF for Agile Software Development 6.2, предназначенный для осуществления гибкого подхода к процессу управления проектами разработки ПО.

3) Microsoft Visual Studio Scrum 2.2., используется небольшими командами (до 10 человек), которые применяют гибкую методологию разработки и терминологию Scrum.

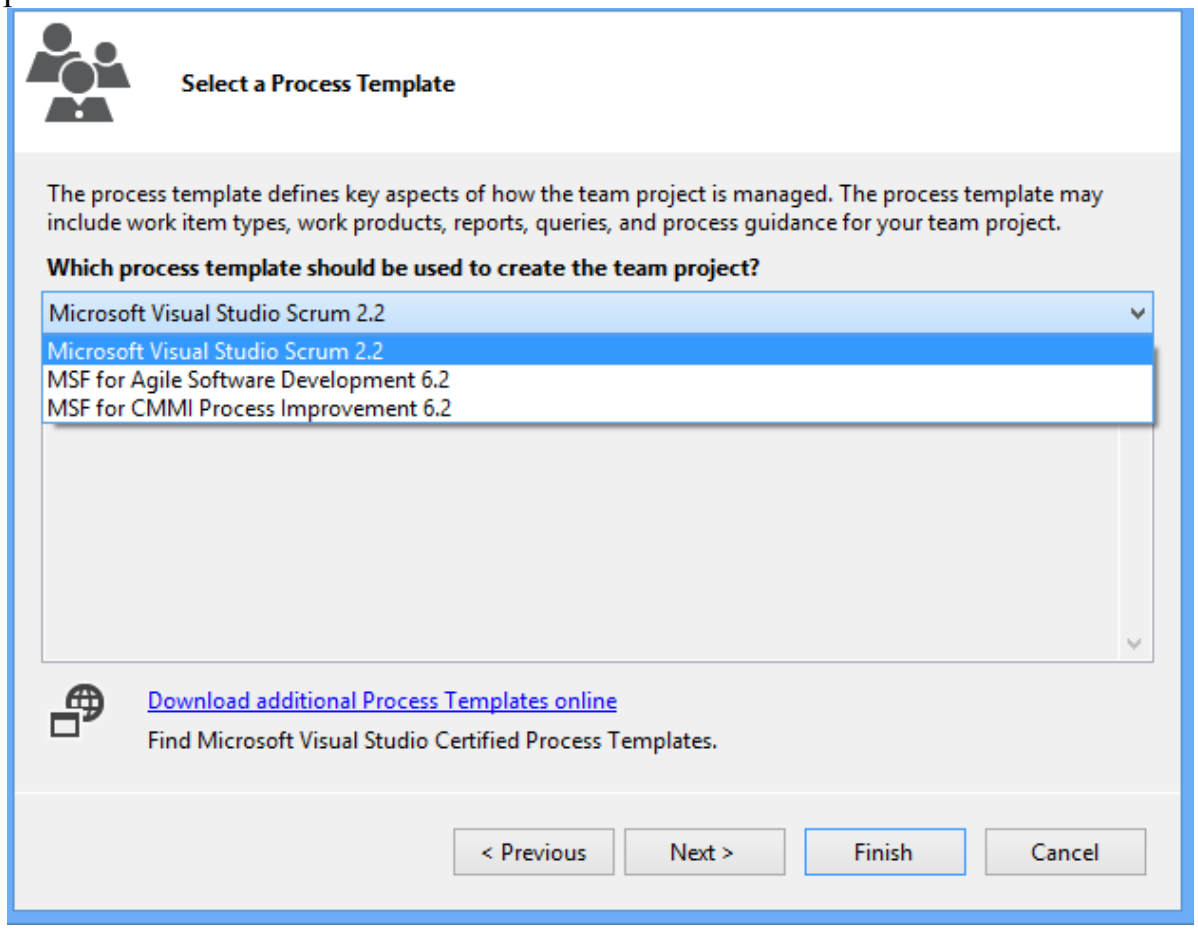

Рис. 4.5. Шаблоны проектов, поддерживаемые TFS 2012

Когда командный проект создается, то появляется возможность произвести настройку и осуществить управление следующими его областями:

1) Рабочими элементами (определяются начальные типы рабочих элементов и на их основе создаются непосредственно сами рабочие элементы, а также общие и пользовательские запросы).

2) Классификациями, которые позволяют задать начальные области и определить итерации проекта.

3) Веб-порталом на основе MS SharePoint, при помощи которого впоследствии можно будет осуществлять управление библиотекой документов проекта.

4) Системой контроля версий, при помощи которой можно задать правила возврата версий, начальные группы безопасности и разрешения на выполнение определенных действий.

5) Отчетами, которые формируются в папки и располагаются на сайте командного проекта.

6) Группами и разрешениями. Сюда входят также группы безопасности TFS и разрешения на выполнение определенных действий, доступных каждому конкретному участнику команды разработчиков.

7) Построениями (определяется стандартный порядок сборки различных версий разрабатываемого ПО).

8) Лабораторией Lab Management (производится ее настройка либо для стандартной среды тестирования, которая может содержать как физические, так и виртуальные машины, либо для среды SCVMM, которая может содержать только виртуальные машины).

9) Управление тестированием, что подразумевает создание стандартной конфигурации тестирования, а также задания состояний разрешения и параметров тестирования и определения переменных тестирования.

Пользовательское видение функциональности разрабатываемой системы (в наиболее благоприятном случае требования технического задания), задачи, которые требуется решить на данном этапе выполнения работ по реализации проекта, ошибки в работе уже написанного программного кода, тестовые случаи, а также препятствия, которые мешают дальнейшему нормальному ходу выполнения работ являются рабочими элементами TFS. Для успешного завершения работ по реализации программного проекта требуется управлять данными элементами, что осуществляется при помощи системы отслеживания рабочих элементов, являющейся частью TFS. Эта система позволяет создать рабочие элементы, следить за их текущим состоянием и хранить историю изменения состояния рабочих элементов. Вся информация о рабочих элементах помещается в БД TFS.

Частью TFS является также централизованная система контроля версий, которая обеспечивает выполнение следующих функций:

1. Осуществлять атомарные возвраты. Все модификации, производимые с файлом, упаковываются в «набор изменений». Возврат измененного файла в таком наборе является транзакцией, производимой с БД TFS. Благодаря атомарным возвратам достигается согласованность базы кода командного проекта.

2. Производить ассоциацию операций возврата с рабочими элементами проекта. Выполнение такой ассоциации позволяет следить за текущим ходом выполнения проекта.

3. Осуществлять объединение и ветвление при реализации программного кода.

4. Производить наборы отложенных изменений, что обеспечивает возможность создания копий программного кода без помещения их в главное хранилище системы.

5. Помечать комплект файлов определенной версии разрабатываемого приложения с помощью текстовой подписи.

6. Производить одновременное редактирование файла несколькими участниками команды, с выполняемым затем объединением всех внесенных изменений.

7. Следить за историей файлов проекта.

8. Определять политику возврата. Благодаря этому можно произвести запуск программного кода, который в свою очередь произведет проверку о допустимости возврата измененных файлов командного проекта.

9. Сгенерировать метаданные о возвращаемых файлах при помощи рабочего элемента, называемого примечание.

10. Использовать прокси-сервер, который позволяет произвести оптимизацию взаимодействия с другими центрами разработки.

Построение решения в TFS осуществляется с помощью сервера Team Foundation Build, что обеспечивает возможность применения стандартной инфраструктуры построения для командного проекта. Существует пять режимов построения:

1. Ручной. В этом режиме имена файлов вручную заносятся в список построения решения.

2. Непрерывной интеграции. В этом случае построение производится при каждом возврате измененных файлов командного проекта.

3. Прокрутки построения. При таком режиме возвращенные модифицированные файлы группируются вместе, поэтому в построение решения включаются только изменения за фиксированный интервал времени.

4. Условного возврата. В этом режиме возвращенные файлы принимают участие в построении решения проекта, если до того они сами успешно прошли построение и слияние с другими файлами командного проекта.

5. Расписания. В этом режиме время построения решения можно назначить на определенные дни недели.

Участники команды разработчиков могут обращаться к сервисам TFS при помощи:

1) Командного обозревателя (Team Explorer) Visual Studio 2012. При его помощи можно осуществлять управление жизненным циклом ПО.

2) Компонента доступа через веб (Team Web Access). В этом случае доступ к функциям управления жизненного цикла ПО осуществляется через веб-интерфейс.

3) Приложения Microsoft Excel, при помощи которого можно задавать и модифицировать рабочие элементы командного проекта массивом, а также создавать отчеты о ходе продвижения работ по реализации проекта на основе запросов о текущем состоянии рабочих элементов.

4) Приложения Microsoft Project, при помощи которого можно задать управление планом проекта, а также ресурсами проекта и задачами проекта, построить календарный план выполнения проекта, сформировать диаграммы Ганта и представления ресурсов.

5) Консоли администрирования TFS, при помощи которой производится настройка работы данного сервера.

6) Командной строки.

7) Средств интеграции, выпущенных другими производителями ПО.

4.2. Организация командной разработки на базе Visual Studio и TFS

В случае командной разработки ПО с использованием инструментальных средств Microsoft необходимо составить календарный план выполнения работ, осуществлять управление рабочими элементами проекта, организовать коммуникацию между участниками команды разработчиков, обеспечить генерацию отчетов, отражающих ход выполнения проекта и возможность их анализа, а также обеспечить возможность внесения необходимых корректив с целью усовершенствования процесса управления проектом. Для решения перечисленных задач в TFS имеются следующие инструментальные средства:

1. Шаблон процесса. Служит для определения стиля командной разработки ПО.

2. Руководство по процессу. В нем содержится описание имеющихся шаблонов процесса разработки ПО.

3. Коллекция командных проектов. Это контейнер, предназначенный для хранения нескольких командных проектов.

4. Командный проект. В нем сохраняются и упорядочиваются данные, содержащие сведения о жизненном цикле разрабатываемого ПО.

5. Компонент, производящий отслеживание текущего состояния рабочих элементов проекта.

6) Портал проекта (панель мониторинга). При помощи данного компонента любой из участников команды разработчиков (при наличии у него соответствующих прав) может получить информацию о текущем состоянии дел проекта.

7. Элементы планирования. Этот компонент позволяет осуществить управление списком требований как на уровне всего проекта, так и на уровне его конкретной итерации.

8. Компонент, позволяющий автоматически сгенерировать отчетность, отражающую состояния разрабатываемого ПО во время его жизненного цикла.

9. Компонент, осуществляющий интеграцию с Microsoft Project и Microsoft Excel. Данный компонент обеспечивает возможность управления процессом разработки из пакета Microsoft Office.

Создание командного проекта является первым этапом разработки ПО с применением инструментальных средств Microsoft. Этот проект должен включать в себя сведения о каждом этапе жизненного цикла разрабатываемого ПО (требования пользователей, предъявляемые к разрабатываемой системе; задачи, которые требуется решить на данной итерации и к моменту завершения работ по реализации проекта; тесты; ошибки при выполнении программного кода; различные версии сборок

разрабатываемого приложения; информацию о причинах, мешающих нормальному ходу работ по реализации проекта).

Когда командный проект создается, то необходимо дать ему имя, создать его описание, выбрать один из используемых шаблонов процесса, произвести настройку работы системы контроля версий, а также определиться с необходимостью создания портала для проекта. При создании командного проекта TFS автоматически генерирует ряд папок с предопределенными названиями, описание функционального назначения которых приведено в табл. 4.1.

Таблица 4.1

Папки, формируемые TFS при создании командного проекта

Название	Описание
Моя работа	Папка, в которой задаются выполняемая и приостановленная работы, доступные рабочие элементы и активные запрошенные задачи анализа кода.
Ожидающие изменения	Папка, в которой обеспечивается возможность сохранения изменений кода в базе данных TFS и извлечения проектных решений для редактирования пользователем.
Рабочие элементы проекта	Папка, в которой хранятся сгруппированные данные о текущих рабочих элементах проекта.
Построения	Папка, в которой формируются и хранятся определения и результаты построения приложений.
Отчеты	Папка, в которой имеются подпапки с отчетами и ссылки на стандартные отчеты.
Документы	Папка, в которой хранятся документы проекта.
Параметры	Папка, в которой хранятся ссылки для перехода к окнам задания параметров проекта (безопасность, состав групп, система управления версиями, области и итерации рабочих элементов, параметры портала и оповещения проекта.

После того как командный проект создан, его руководитель начинает заниматься формированием команды разработчиков. Для каждого из участников команды разработчиков руководитель определяет доступ к проекту, как в целом, так и к его конкретным артефактам.

Для того чтобы структурировать проект применяются области и итерации. Области можно задать исходя из определенного функционального назначения этапа работ, а итерации определить как определенную совокупность работ, выполняемую на фиксированном интервале времени. При использовании методологии MSF и ее реализации с помощью инструментальных средств Microsoft в качестве ключевых сущностей командного проекта выступают рабочие элементы. Наименование и набор рабочих элементов в разных шаблонах процессов командной разработки несколько отличается друг от друга.

Например, для шаблона процесса, использующего методологию Agile при командной работе по реализации проекта рабочими элементами будут:

1. Пользовательское описание функциональности (User Story).
2. Задача (Task).
3. Ошибка (Bug).
4. Препятствие (Issue).
5. Тестовый случай.

Рабочий элемент «User Story» определяет требования пользователей к функциональности создаваемой системы. Рабочий элемент «Task» применяется в проекте для распределения работ между участниками команды и контроля за ходом их выполнения. Одной из функций задач является задание плана реализации пользовательских требований. Рабочий элемент «Bug» используется для фиксации некорректного поведения созданного программного кода. Рабочий элемент «Issue» применяется для отслеживания объектов или событий, которые препятствуют нормальному выполнению хода работ по реализации проекта. Данные объекты или события подлежат устранению в ходе выполнения текущей или следующей итерации. Рабочий элемент «Тестовый случай» предназначен для выполнения верификации разработанного ПО.

В информацию о рабочем элементе проекта могут быть помещены сведения об имени, описание назначения, текущее состояние, принадлежность к той или иной рабочей области проекта. Описать рабочие элементы командного проекта можно при помощи:

1) Командного обозревателя в Visual Studio.
2) Microsoft Project.
3) Microsoft Excel.

Несколько рабочих элементов «User Story» могут логически связываться между собой, а также с другими рабочими элементами проекта, которые в этом случае выступают в качестве родительских или дочерних элементов. Например, в качестве дочерних элементов для «User Story» могут выступать задачи или ошибки.

Задачи распределяются среди разработчиков ПО. Программисты осуществляют кодирование и выполняют модульное тестирование задач в IDE Visual Studio. Разрабатываемый программный код помещается в хранилище кода, где контроль над ним осуществляется при помощи системы управления версиями. Построение решения на основе работоспособных кодов задач осуществляется при помощи Team Service Build, после чего они помещаются в систему управления версиями TFS. В ходе выполнения работ по написанию ПО разработчик выполняет извлечение (checking out) файлов, содержащих программные коды задач приложения в рабочую область на своем ПК. После внесения в полученный программный код всех необходимых изменений разработчик должен осуществить его возврат (checking in) в хранилище кода TFS. При выполнении процедуры возврата кода буде сгенерирован набор изменений (change set), в котором отражается вся информация, связанная с возвратом (дата и время модификации, исправления, данные о владельце, ссылки

на рабочие элементы, примечания и т.д.). Это позволяет разработчикам ПО иметь доступ к различным версиям файлов и производить анализ, внесенных в них изменений.

Тестировщики на базе тестовых случаев осуществляют разработку тестов, при помощи которых можно проверить соответствие работы системы заданным пользовательским требованиям. С этой целью имеющиеся файлы программного кода извлекаются из хранилища TFS и подвергаются тестированию. При обнаружении ошибки будет сформирован рабочий элемент «Bug», который отправляется разработчику для исправления. Прохождение всех тестов означает, что пользовательское требование выполнено и для него производится установка состояния «Готово».

В TFS 2012 встроена мощная система, обеспечивающая автоматический сбор информации о ходе выполнения проекта и производящая на основе этой информации генерацию отчетов. Система позволяет сгенерировать отчеты различного вида, предназначенные как для руководителя, так и для обычных участников команды разработчиков. Сгенерированные отчеты помещаются в хранилище данных TFS. Благодаря отчетам появляется возможность наблюдения за метриками проекта. В TFS имеется готовый набор шаблонов отчетов, но также существует возможность создания своих собственных отчетов.

В TFS информация сохраняется при помощи следующих объектов:

1) Операционное хранилище.

2) Хранилище данных.

3) OLAP-куб.

Операционное хранилище TFS составляют несколько реляционных БД, предназначенных для хранения такой информации, как исходные тексты программного кода, отчеты о выполнении построения решения, результаты прохождения тестов, текущее состояния рабочих элементов проекта. Информация из операционного хранилища через заданный интервал времени перемещается в хранилище данных. Информация, размещенная в хранилище данных, используется для выполнения запросов и построения на основе этих результатов этих запросов различного вида отчетов.

OLAP-куб TFS представляет из себя многомерную БД, в которой хранятся агрегированные данные по проекту, использующиеся для создания аналитических отчетов. В OLAP-куб информация копируется из хранилища данных через заданные интервалы времени. Получить доступ к информации, размещенной в OLAP-кубе можно из различных клиентских приложений, например, конструктора SQL-отчетов или Microsoft Excel. В TFS изначально два набора шаблонов отчетов, которые могут быть использованы клиентскими приложениями: Microsoft Excel Reports и Reporting Services Reports. Приложение Microsoft Excel позволяет сформировать отчеты из данных OLAP-куба TFS, либо используя результаты ответов на запросы рабочих элементов проекта. При необходимости сгенерированные отчеты можно разместить на SharePoint-портале проекта.

Для создания командного проекта необходимо выполнить следующие шаги для TFS:

1. Запустить консоль администрирования TFS 2012 (рис. 4.6).

2. На вкладке «Общие» выбрать пункт «Администрирование безопасности» (рис. 4.6).

3. Добавить пользователей и наделить их необходимыми правами доступа в соответствии с их ролями в процессе разработки. (рис. 4.7)

3.1) установить переключатель на «Пользователь или группа Windows».

3.2) ввести доменное имя пользователя (если командный проект создается в рамках домена) и нажать кнопку «Ок».

Для присоединения локальной машины к командному проекту необходимо выполнить следующие шаги:

1) Если обращение к серверу TFS происходит через (VPN – виртулаьную частную сеть), организованную, например, с помощью TeamViewer, то необходимо скорректировать файл hosts (его расположение: Windows\System32\drivers\etc\hosts) записав в него строку вида:

<ip-адрес партнера TeamViewer> <имя сервера>

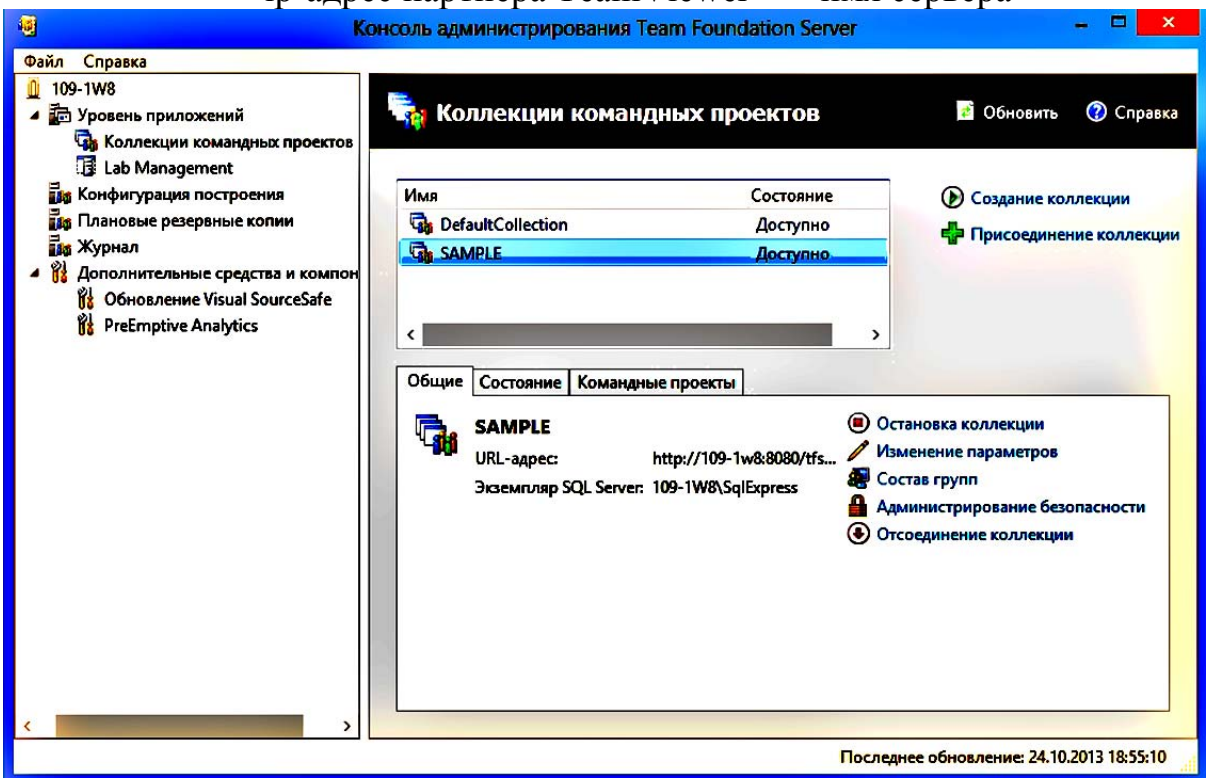

Рис. 4.6. Внешний вид консоли администрирования TFS 2012

2) Запустить Visual Studio 2012.

3) В меню выбрать вкалдку Team, а в ней пункт “Connect to Team Foundation Server” (рис. 4.8).

4) Выбрать из выпадающего списка имя требуемого сервера. Если его в списке нет, нажать кнопку Servers, в открывшемся окне нажать кнопку Add, ввести имя сервера (рис. 4.9).

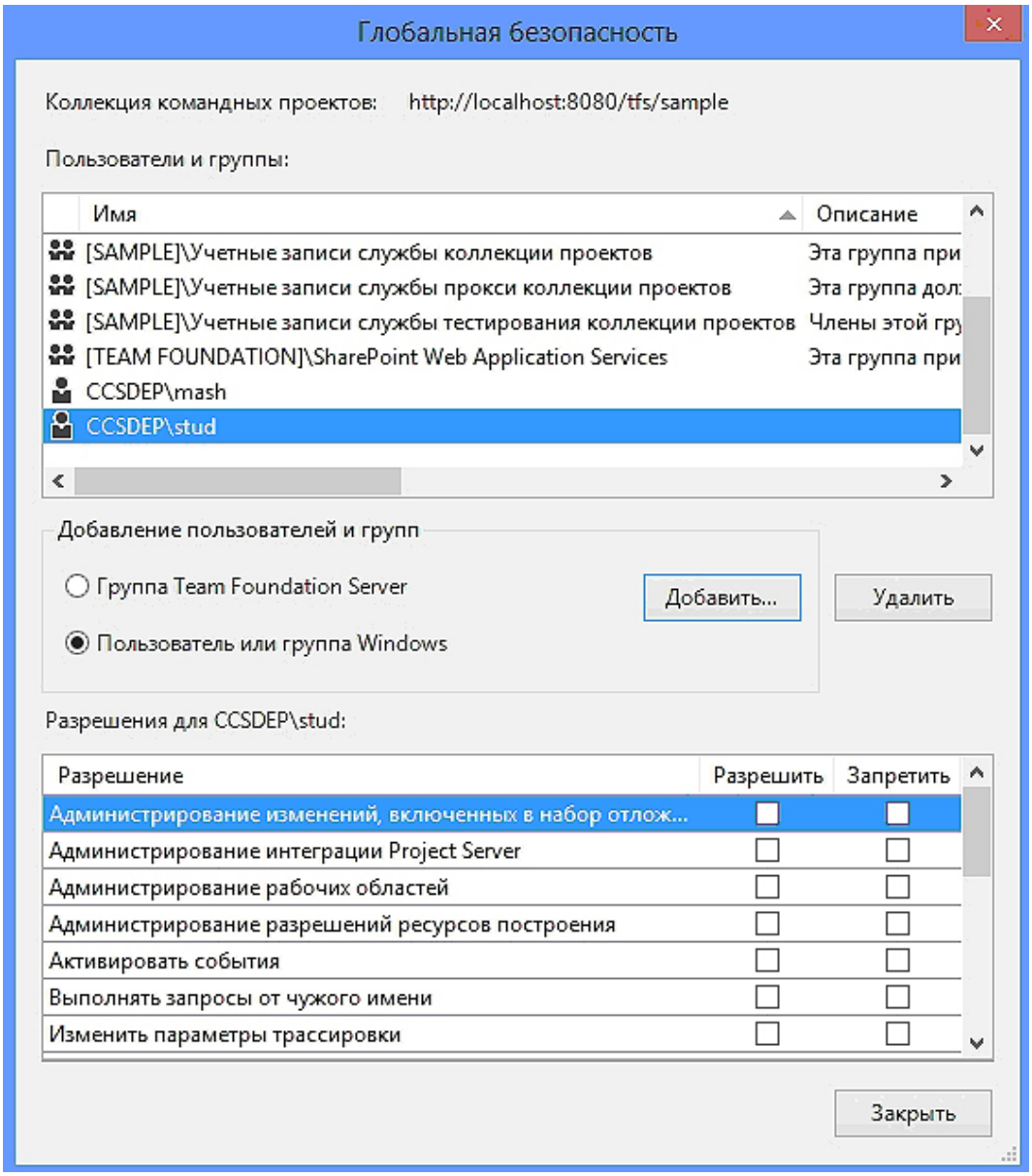

Рис. 4.7. Добавление в командный проект TFS 2012 пользователей и групп и наделение их правами доступа

5) В списке Team Project Collections выбрать коллекцию проектов, в списке Team Projects – командные проекты (рис. 4.10). Нажать кнопку «Connect».

6) Для создания нового командного проекта в Team Explorer выбрать вкладку «Create New Team Project» (рис. 4.11)

7) Ввести имя нового проекта (обязательно) и его описание (не обязательно) (рис. 4.12). Нажать кнопку «Next».

8) Выбрать один из используемых шаблонов командной разработки, поставляемой вместе с VS2012 (рис. 4.5).

9) Создать пустую папку системы управления версиями (пункт «Create an empty source control folder») (рис. 4.13)

10) Проверить правильность всех введенных данных (рис. 4.14) и нажать кнопку «Finish».

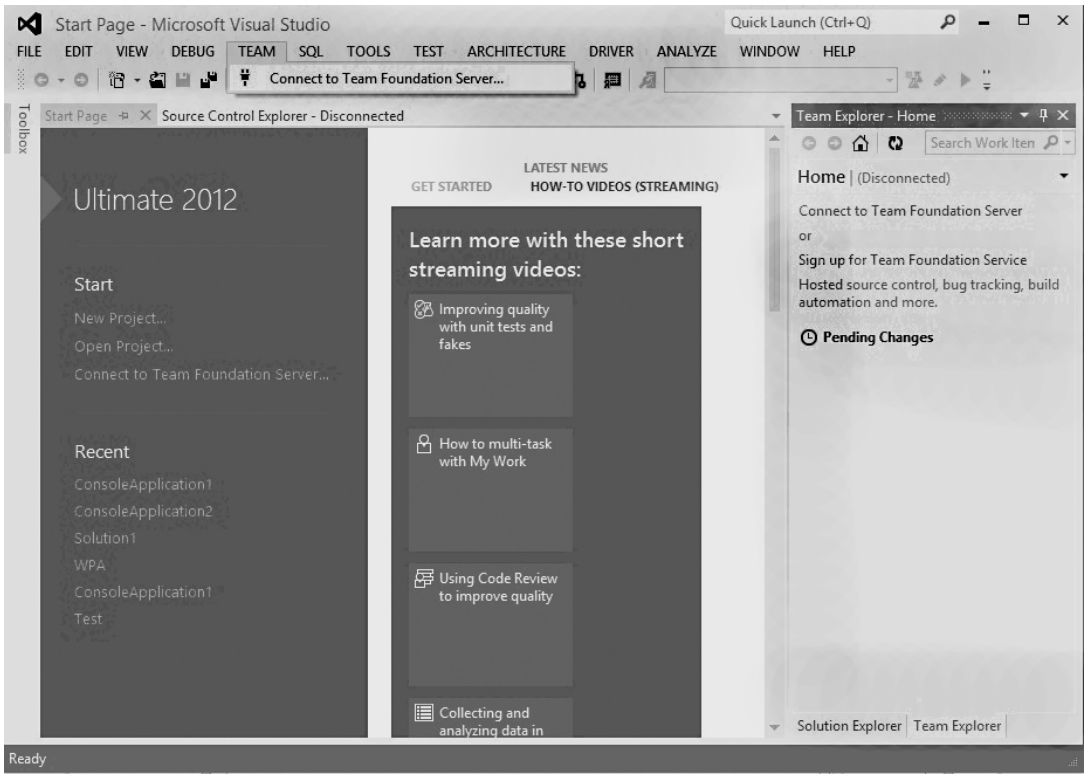

Рис. 4.8. Присоединение к командному проекту из VS2012

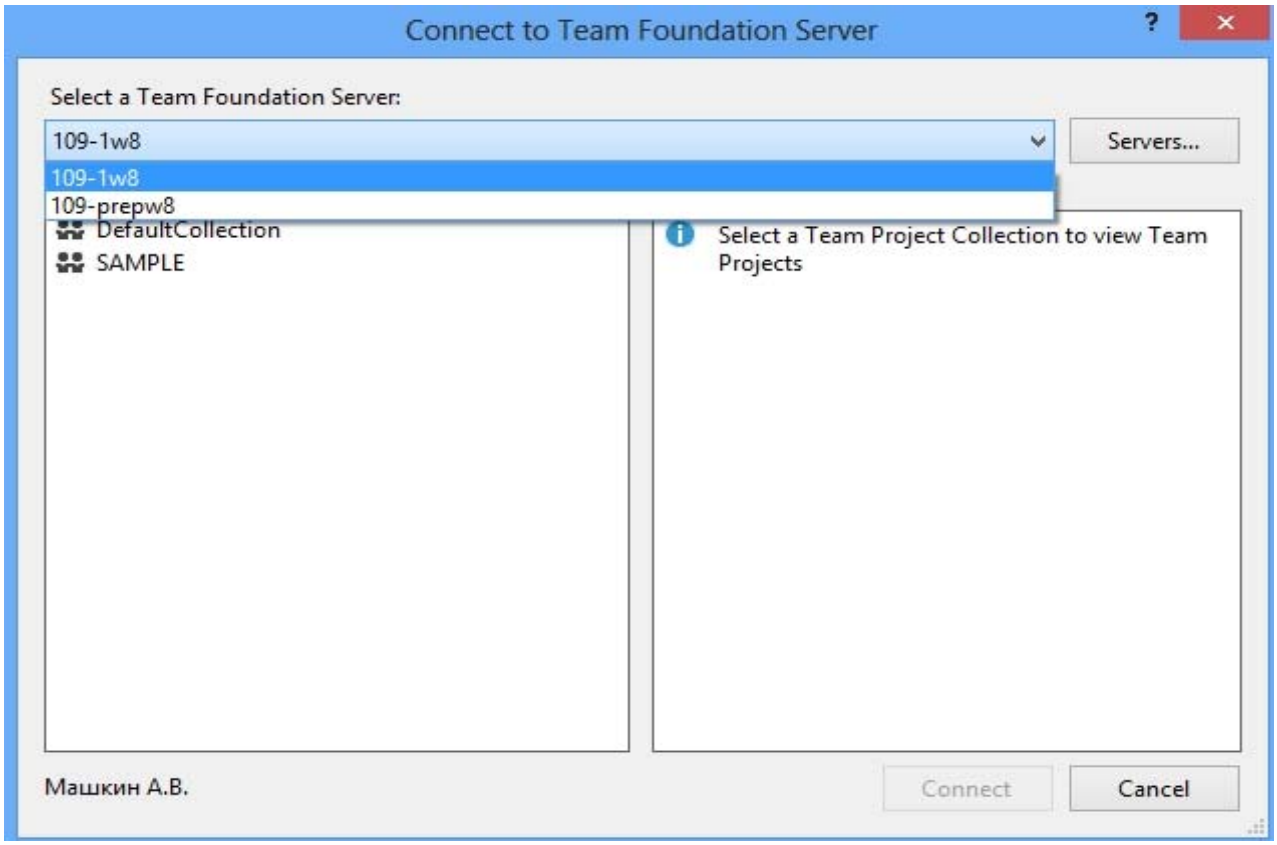

Рис. 4.9. Выбор требуемого сервера TFS

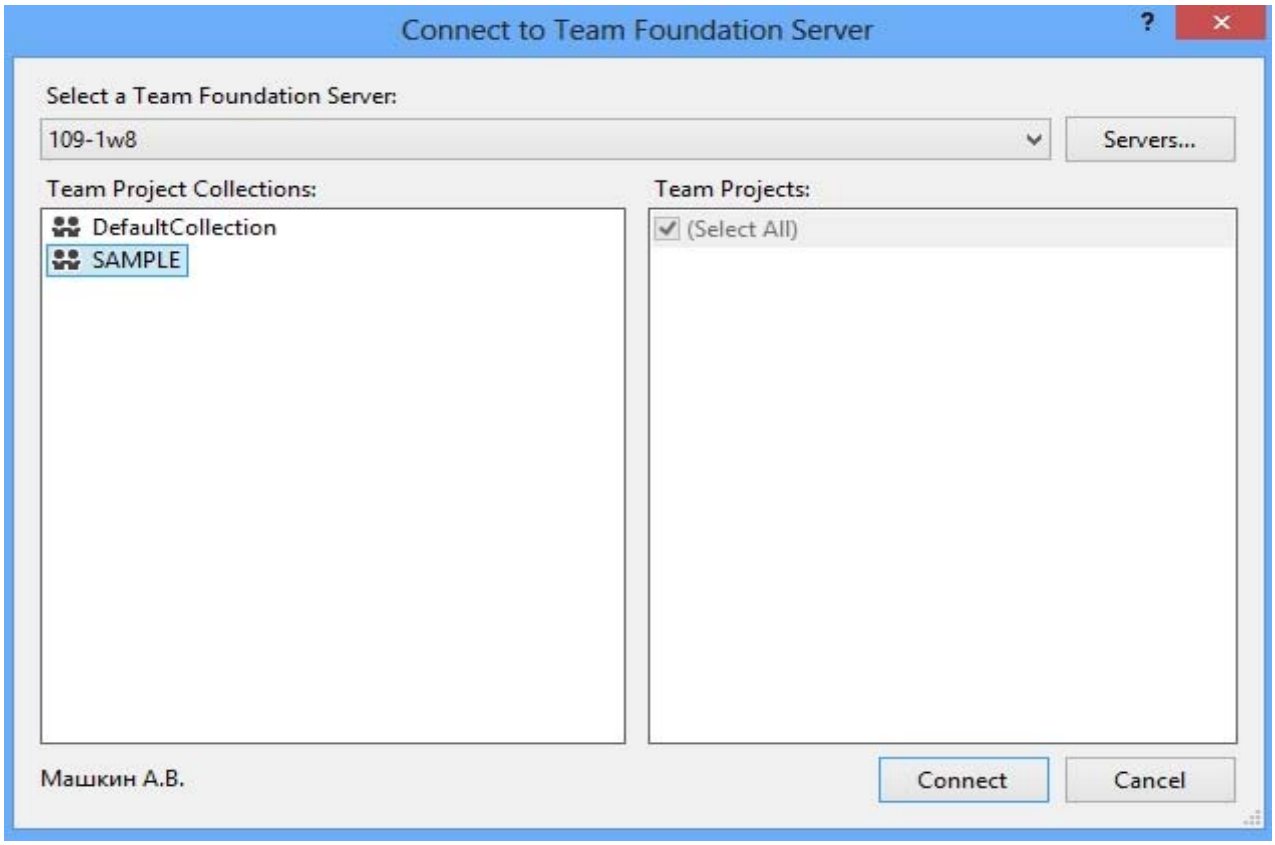

Рис. 4.10. Выбор коллекции проектов

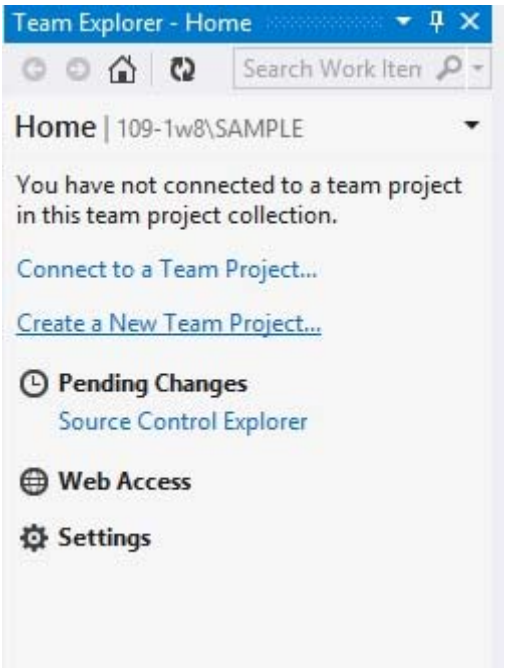

Рис. 4.11. Создание нового командного проекта

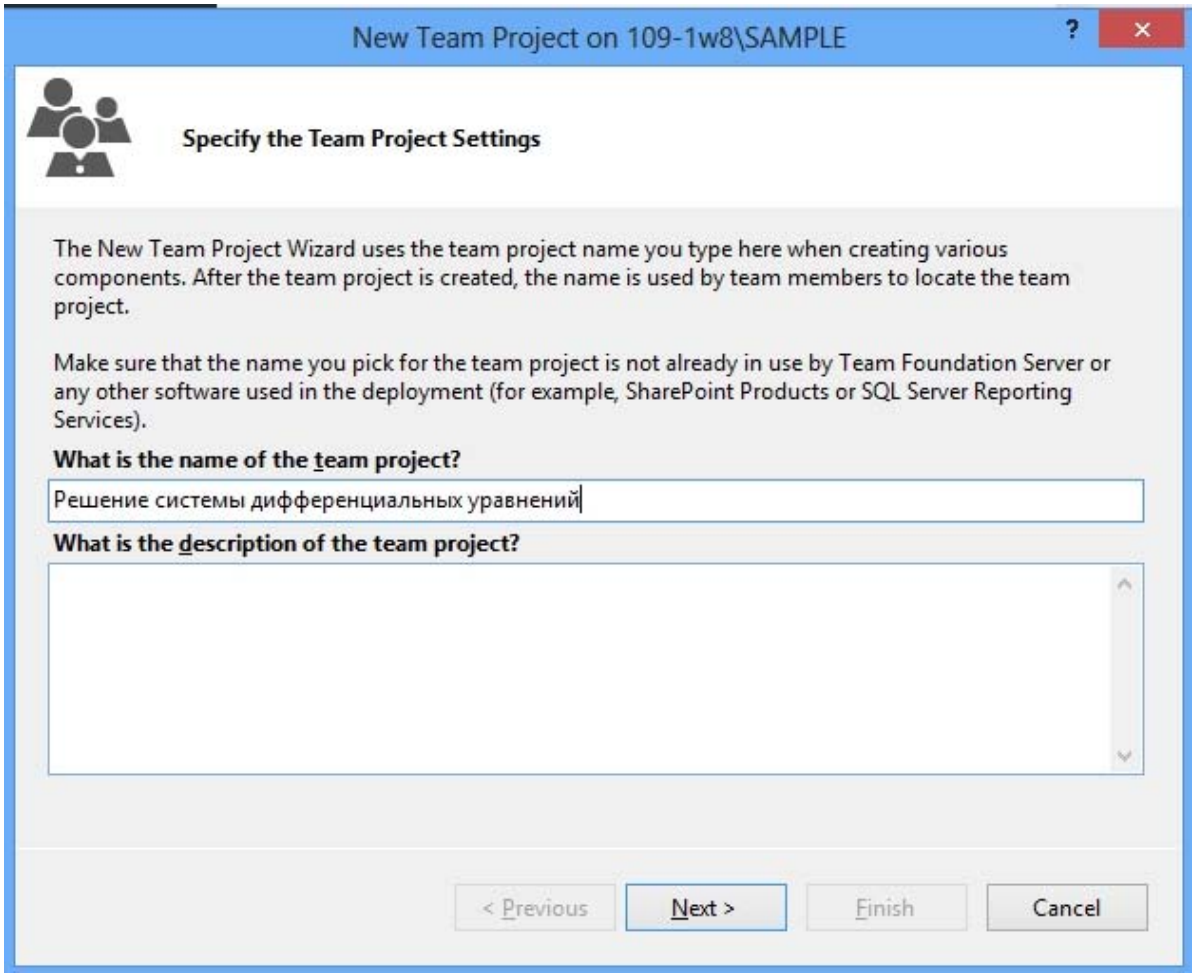

Рис. 4.12. Создание имени и описание проекта

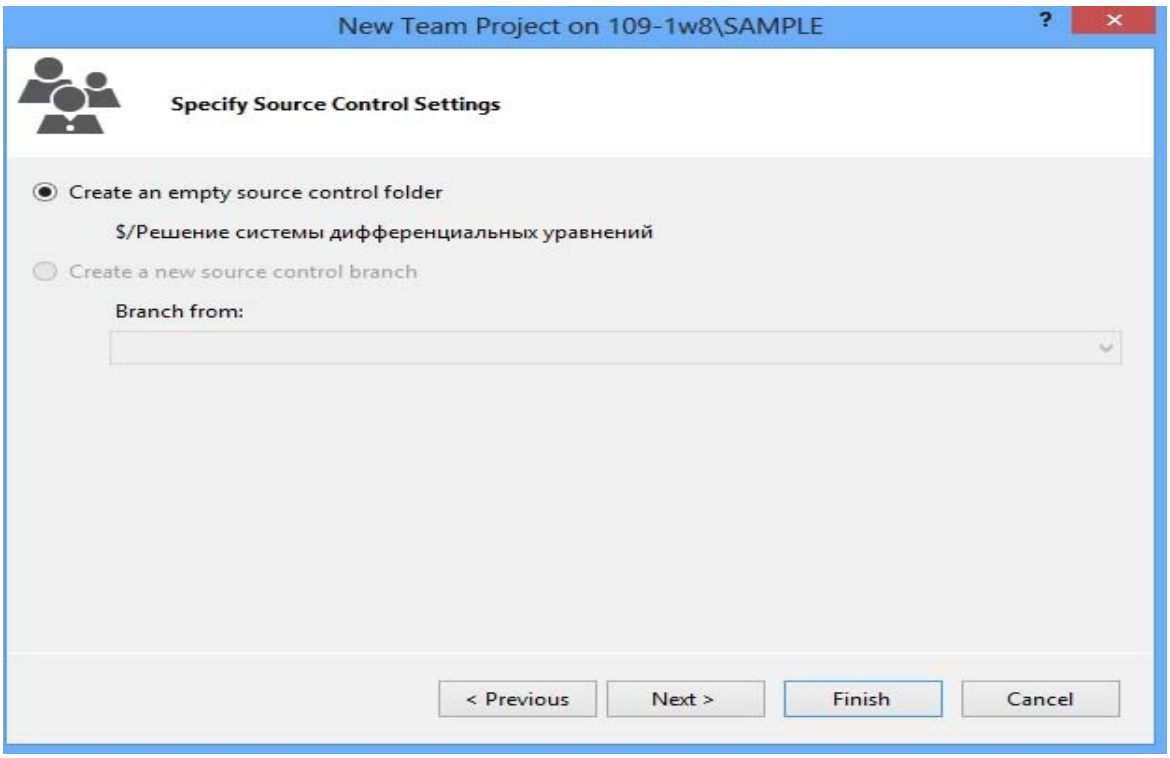

Рис. 4.13. Создание папки для системы управления версиями

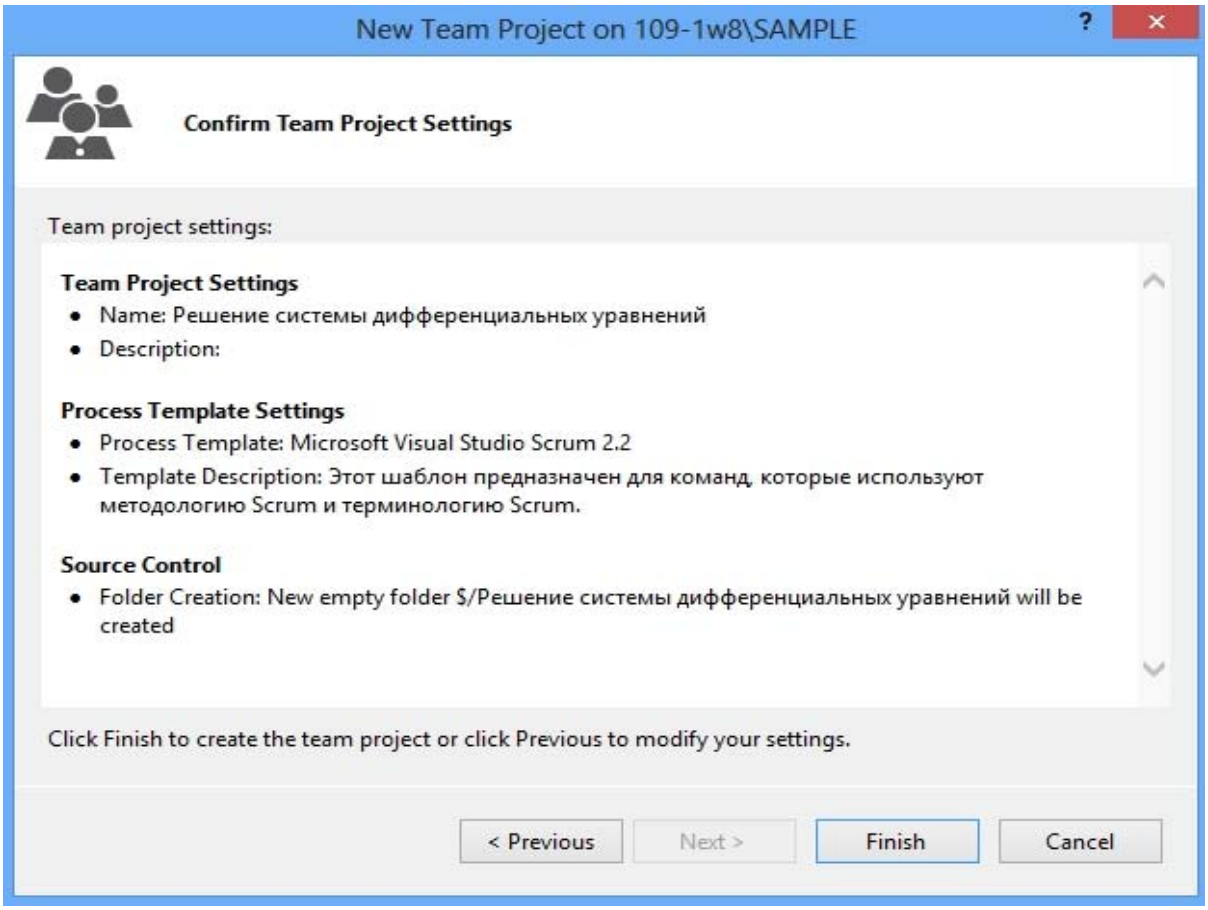

Рис. 4.14. Проверка введенных данных

11) После завершения создания проекта нажать кнопку «Close».

12) В Team Explorer на вкладке Home выбрать пункт Settings, а нем выбрать вкладку Group Membership (роли в группах) (рис. 4.15).

13) В списке команд выбрать команду созданного проекта. В правом окне нажать кнопку и выбрать пункт «Добавление пользователя или группы Windows» (рис. 4.16).

14) Выбрать из списка пользователей домена, которых необходимо включить в команду. Нажать кнопку «Сохранить изменения» (рис. 4.17).

15) В Team Explorer на вкладке Home выбрать пункт Source Control Explorer (если он не открыт).

16) Щелкнуть правой кнопкой мыши по папке командного проекта, в контекстном меню выбрать Advanced – Map to Local Folder (рис. 4.18).

17) В поле Local Folder ввести путь к локальной папке, в которой будет храниться локальная копия командного проекта.

18) Создать новый проект (File – New Project) любого типа.

19) Отредактировать проект нужным образом.

20) Сохранить изменения.

21) В Solution Explorer щелкнуть правой кнопкой мыши по проекту, в контекстном меню выбрать Add Solution to Source Control (рис. 4.19).

22) В открывшемся окне выбрать корень папки, где будет располагаться проект и нажать ОК.

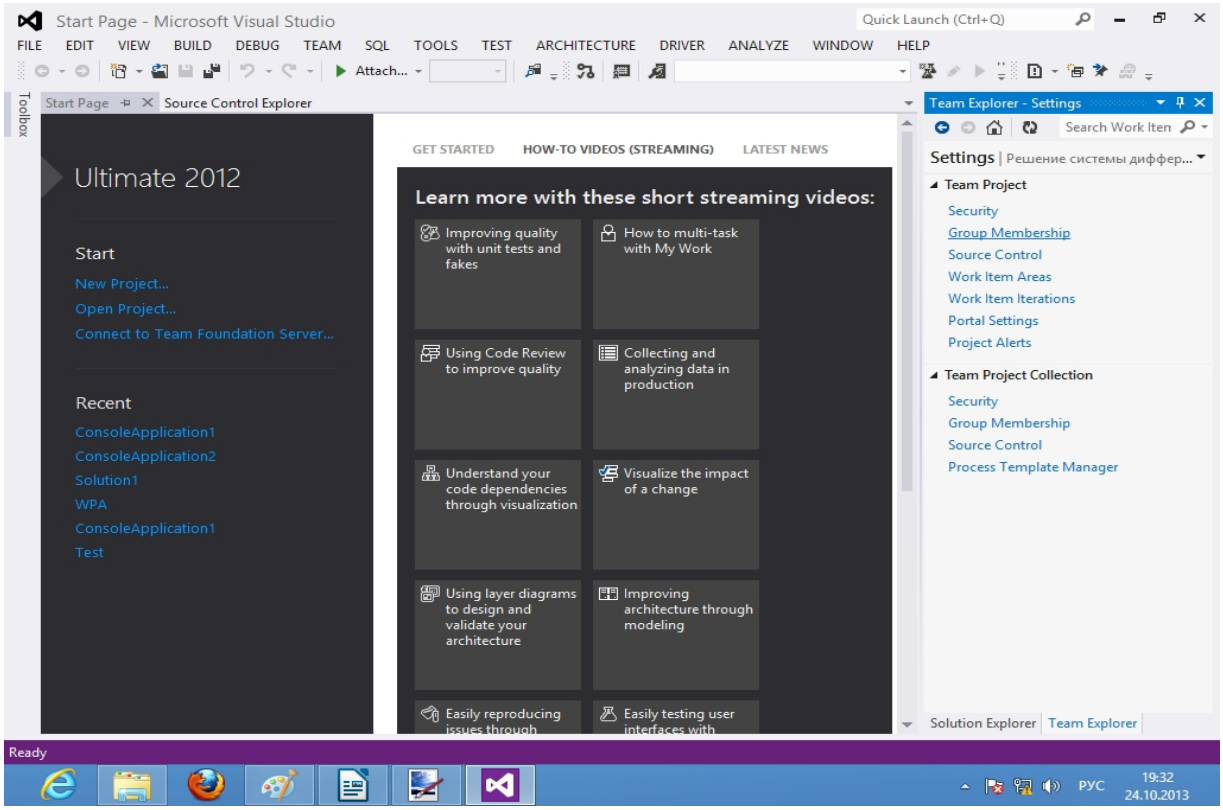

Рис. 4.15. Переход на вкладку с определением ролей в группе

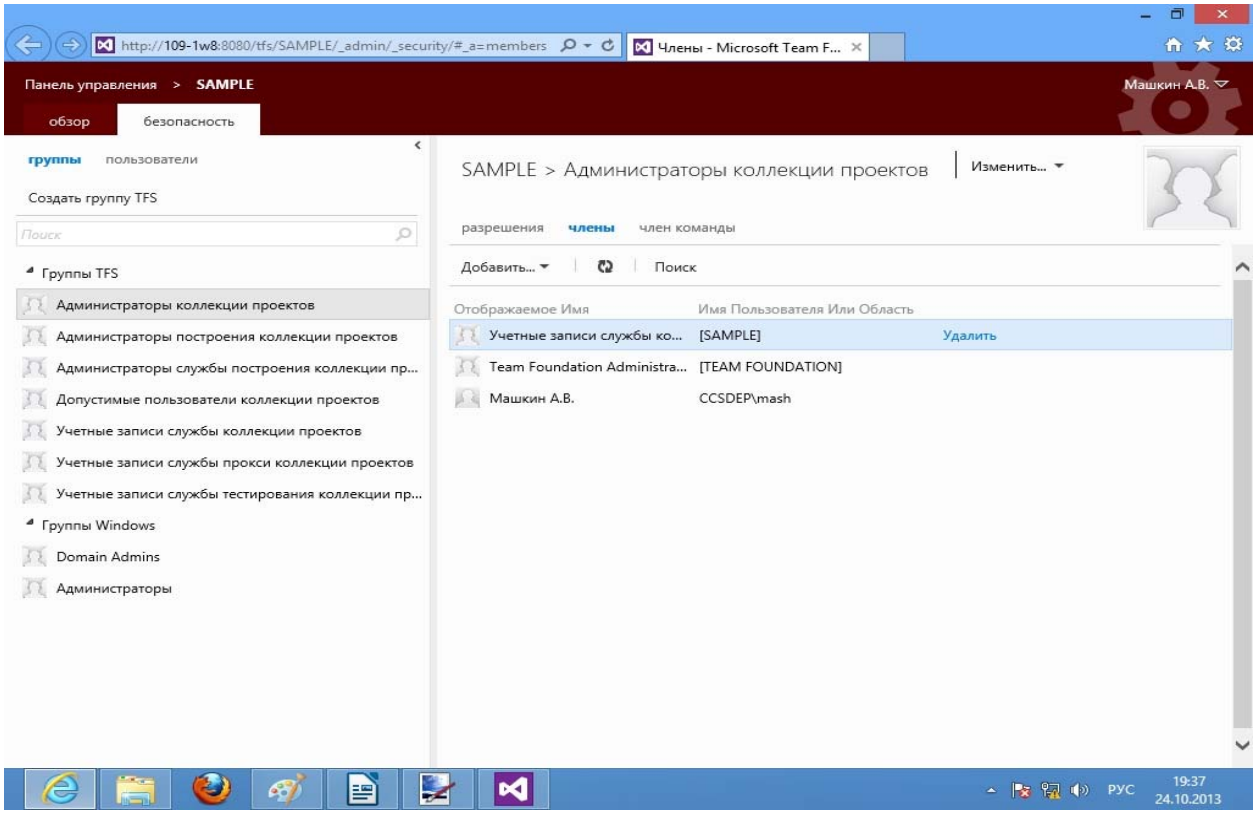

Рис. 4.16. Вкладка администрирование коллекции проектов

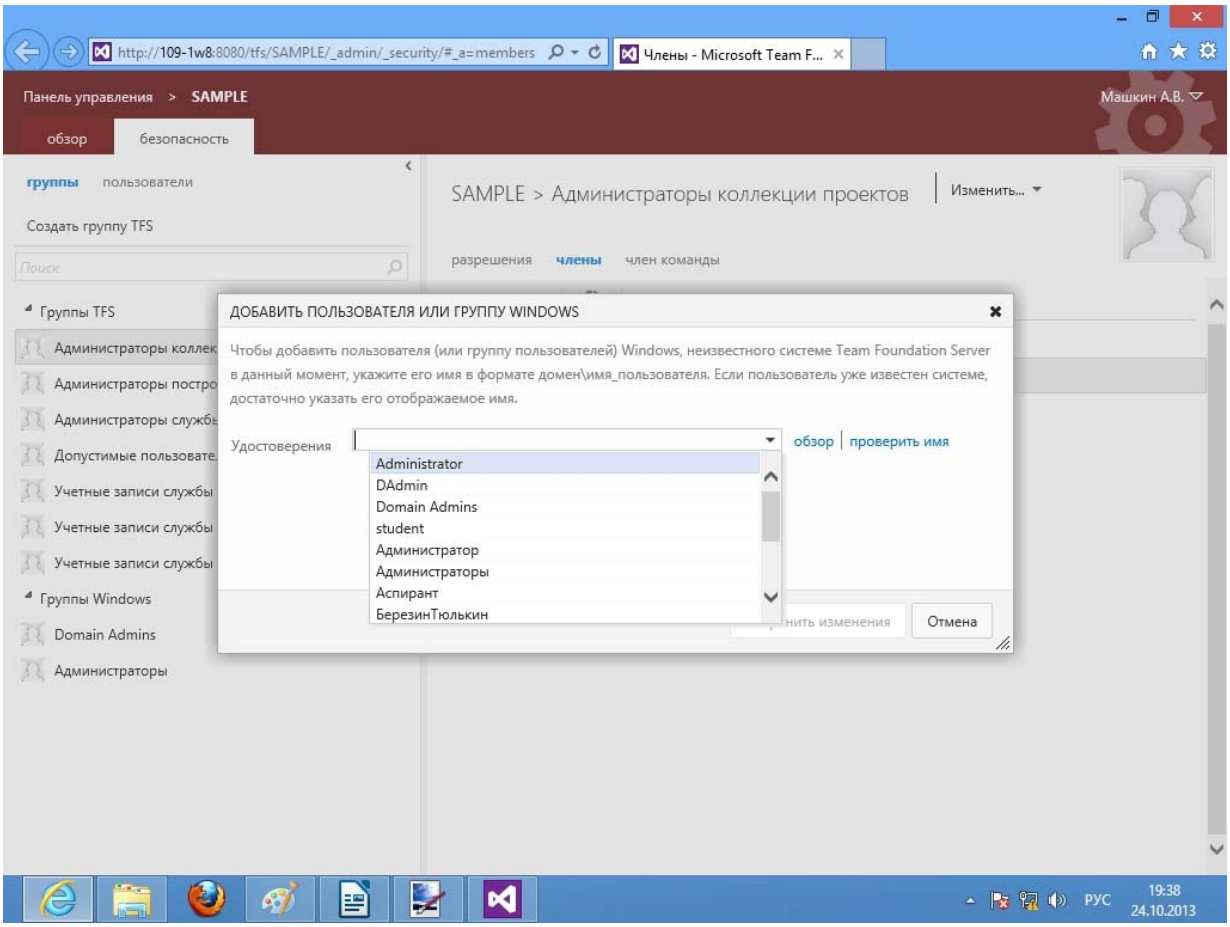

Рис. 4.17. Добавление пользователей или групп Windows в командный проект

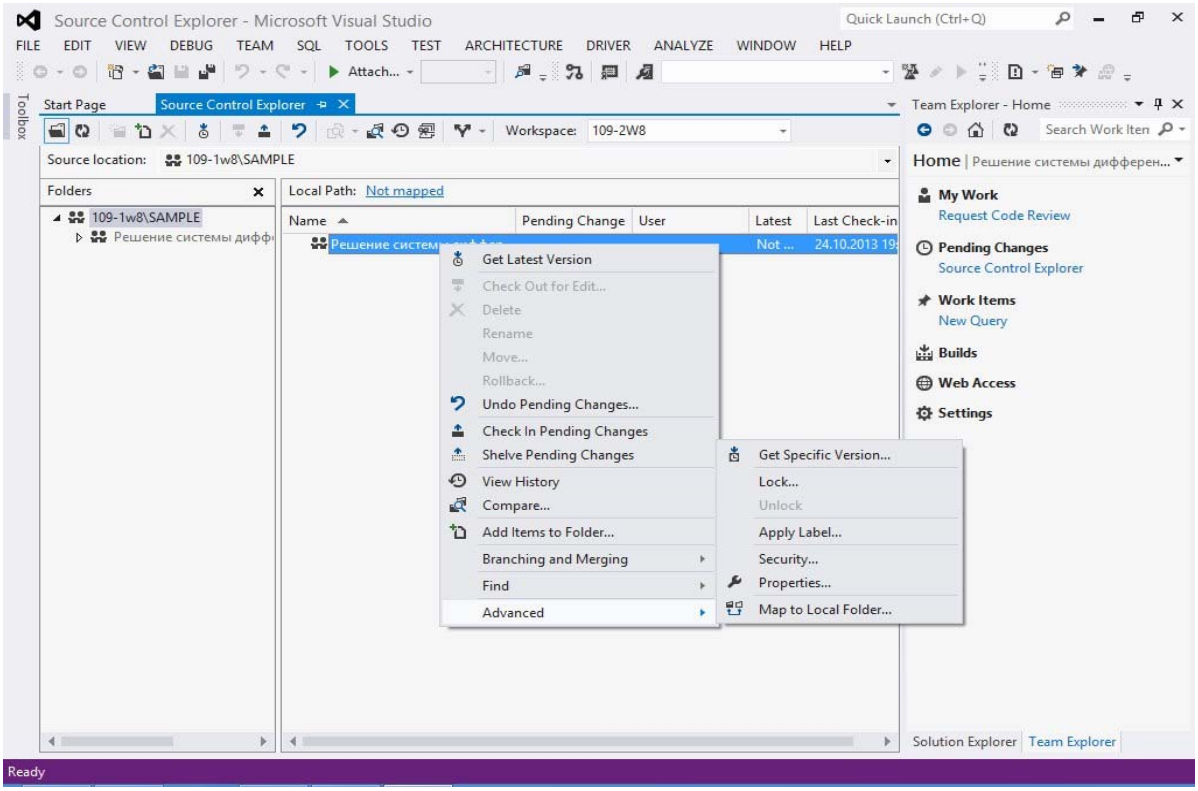

Рис. 4.18. Создание папки на локальной машине для хранения содержимого командного проекта

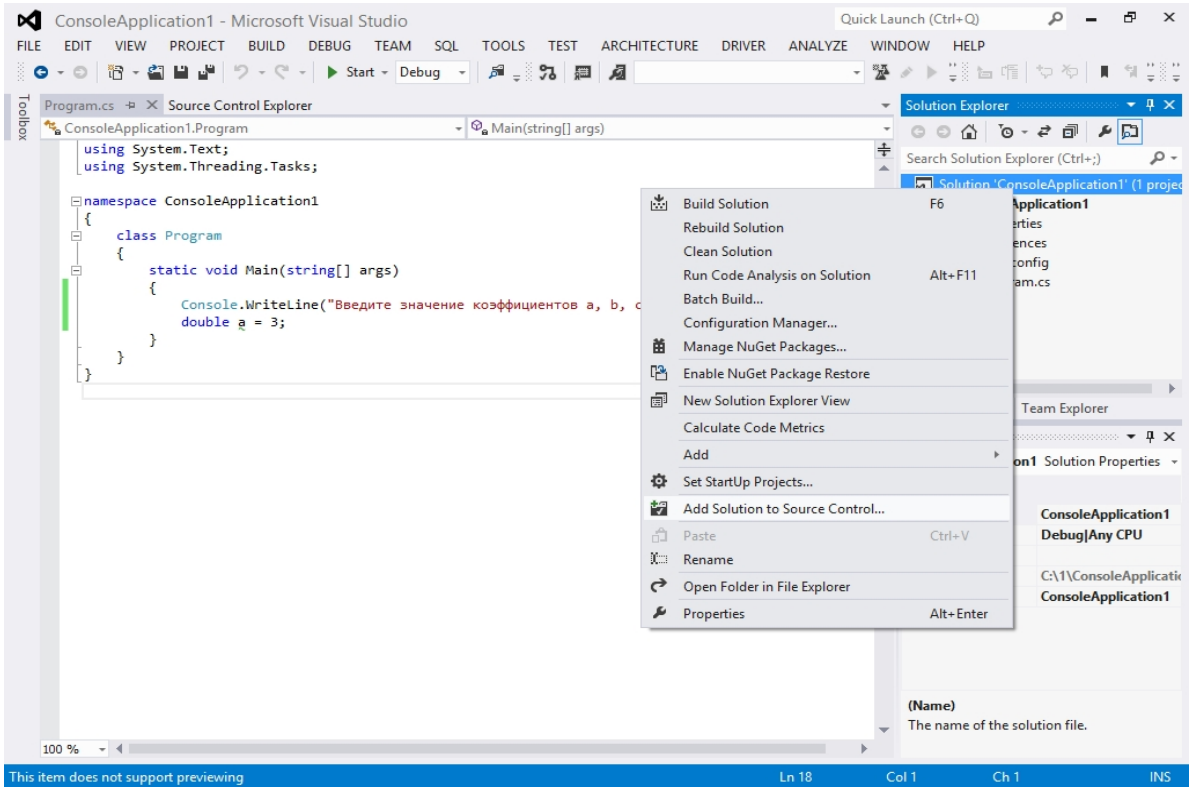

Рис. 4.19. Добавление контроля версий в командный проект

23) В Solution Explorer щелкнуть правой кнопкой мыши по проекту, в контекстном меню выбрать Chek In (рис. 4.20).

24) В открывшемся окне Team Explorer на вкладке Pending Changes нажать кнопку Check In.

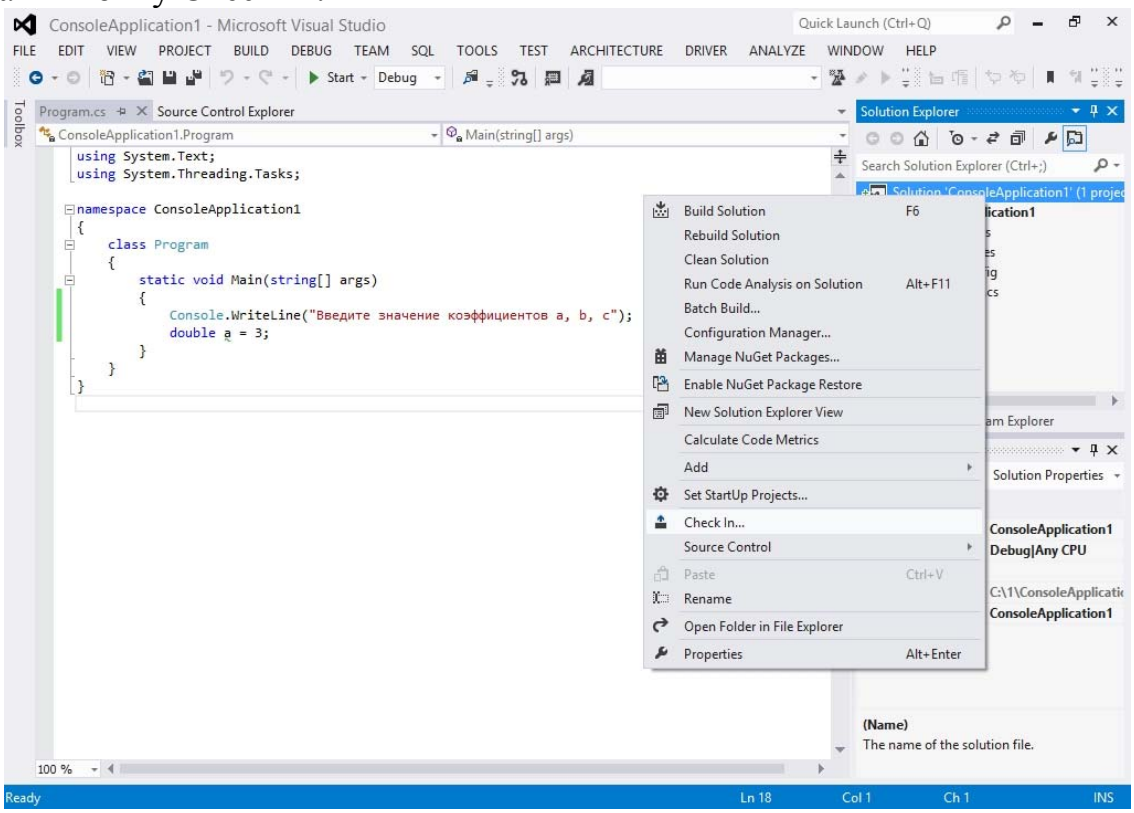

Рис. 4.20. Фиксирование изменений в командном проекте

На другой локальной машине, присоединяемой к командному проекту, необходимо выполнить следующие шаги:

1) Повторить пункты 1-4, а также 14-16.

2) В дереве Source Control Explorer выбрать папку созданного на другой машине проекта, щелкнуть по ней правой кнопкой мыши и выбрать Get Latest Version.

3) Открыть существующий проект (File – Open) из папки, к которой привязана папка командного проекта.

4.3. Обеспечение качества разрабатываемого ПО

Согласно методологии MSF существует несколько критериев, при помощи которых можно выполнить оценку качества разработанного ПО. Качественное ПО должно соответствовать всем требованиям технического задания (функциональным и не функциональным) с учетом их возможного изменения и дополнения в ходе реализации проекта, отвечать ожиданиям пользователя и представлять ценность для бизнеса. Поэтому качество разрабатываемого ПО должно контролироваться на протяжении всего жизненного цикла программного продукта. Качество разрабатываемого ПО начинает формироваться с формулирования необходимых требований, предъявляемых к разрабатываемой системе. При этом требуется определить функции, которые будет реализовывать разрабатываемая система, а также методы, при помощи которых будет производится проверка функциональности разработанного ПО.

Вне зависимости от сферы применения (наука и образование, бизнес, медицина, социальная сфера, веб и т.д.) разработанное ПО должно обладать высоким качеством. Качественное ПО должно обеспечивать удовлетворение определенного уровня потребностей своего конечного пользователя. Качественно созданное ПО должно отвечать также и нефункциональным требованиям, предъявляемым к разрабатываемой системе, например таким как: надежность, удобство использования и сопровождения, производительность, безопасность, защищенность. Надежность – это способность программы выполнять заданные функции в предопределенных условиях в течение фиксированного промежутка времени. Удобство сопровождения задает легкость, с которой обслуживается разработанное ПО в плане устранения обнаруженных недостатков, модификации с целью приведения разработанной системы в соответствии с дополнительными требованиями и т.д. Производительность ПО обычно определяют затратами времени на исполнение длительных операций или заданных транзакций. Безопасность – это невозможность выполнения программой несанкционированных действий. Защищенность задает уровень безопасности системы (возможность блокировки от несанкционированного доступа, предотвращение потери работоспособности системы и т.д.).

Реализованное в инструментальных средствах Microsoft управление жизненным циклам ПО предотвращает лишние временные затраты на

повторное проектирование архитектуры разрабатываемой системы и ее модификацию, т.е. изначально обеспечивается возможность сразу же приступить к разработке высококачественного ПО.

Тестирование, производимое на протяжении всех итерационных циклов разработки, служит для подтверждения, что параметры частей разрабатываемой системы соответствуют заданным критериям ее качества. Основной задачей тестирования является как можно более раннее выявление отклонения функционирования системы от требований, сформулированных в техническом задании или пользовательской истории (user story). В этом случае можно минимизировать затраты на исправление допущенных ошибок.

На всем этапе разработки жизненного цикла ПО используются различного рода тесты (автоматические и ручные), которые позволяют гарантировать, что промежуточные версии разрабатываемой системы удовлетворяют заданным показателям качества. Применяемые при использовании методологии MSF виды тестирования и области их использования приведены в табл. 4.2.

Таблица 4.2

Виды тестирования

Вид тестирования	Сфера применения
Модульное тестирование	Предназначено для проверки правильности функционирования методов классов ПО. Модульные тесты пишутся и исполняются разработчиками в процессе написания кода. Модульное тестирование применяется как для проверки качества кода приложения, так и для проверки объектов баз данных.
Исследовательское тестирование	Предназначено для тестирования, при котором тестировщик не имеет заранее определенных тестовых сценариев и пытается интуитивно исследовать возможности программного продукта и обнаружить и зафиксировать неизвестные ошибки.
Интеграционное тестирование	Используется для проверки корректности совместной работы компонентов программного продукта.
Функциональное тестирование	Предполагает проверку конкретных требований к ПО и проводится после добавление к системе новых функций.
Нагрузочное тестирование	Предназначено для проверки работоспособности программного продукта при предельной входной нагрузке.
Регрессионное тестирование	Применяется при внесении изменений в программное обеспечение с целью проверки корректности работы компонентов системы, которые потенциально могут взаимодействовать с измененным компонентом.
Комплексное тестирование	Предназначено для тестирования функциональных и нефункциональных требований всей системы программного продукта.
Приемочное тестирование	Представляет собой функциональные испытания, которые должны подтвердить то, что программный продукт соответствует требованиям и ожиданиям пользователей и заказчиков. Приемочные тесты пишутся бизнес-аналитиками, специалистами по контролю качества и тестировщиками.

Модульные, нагрузочные и тесты пользовательского интерфейса можно создавать в IDE Visual Studio 2012 версии Ultimate. Вместе с VS2012 поставляются следующие шаблоны тестов (тестовых проектов):

1) Проект модульного теста, который позволяет создавать модульные тесты в процессе разработки.

2) Проект с веб-тестами производительности и нагрузочными тестами.

3) Проект с закодированными тестами пользовательского интерфейса.

Для организации работы тестировщиков в VS2012 применяется компонент Microsoft Test Manager (MTM). MTM обеспечивает возможность тестирования на всем этапе разработки жизненного цикла ПО. Он позволяет произвести планирование тестирования, непосредственно выполнить тесты и собрать информацию о результатах тестов. Система MTM интегрирована с TFS. При задании плана тестирования в него заносятся требуемые наборы тестов, модели тестовых случаев и конфигурация среды, необходимая для проведения тестирования. С помощью MTM можно выполнять автоматическое, ручное и исследовательское тестирование. Результаты выполнения тестов сохраняются в БД, и на их основе в дальнейшем можно генерировать различные автоматические отчеты. Ошибки в работе создаваемой системы, обнаруженные во время исполнения тестов, фиксируются, документируются, и отчеты по ним передаются разработчикам. Разработчики для устранения обнаруженных ошибок выполняют модификацию существующего у них программного кода, тем самым вызывая необходимость применения регрессионного (повторного) тестирования. Система MTM автоматически создает план регрессионного тестирования, помещая туда тесты, подлежащее повторной прогонке.

Для разработчиков и тестировщиков ПО также предназначен имеющийся в VS2012 диспетчер виртуальной среды Lab Management, который позволяет достаточно адекватно эмулировать реальную среду, в которой, как предполагается, будет функционировать разработанное ПО. Такие модели среды выполнения могут применяться для прогона автоматических тестов и исполнения разработанного ПО.

Качество разработанного ПО во многом зависит от того, легко или трудно выполнять модификацию разработанного кода, а также определяется тем насколько созданный программный код легок для понимания. Разработанное ПО может реализовывать всю требуемую функциональность, но его модификация может вызывать затруднения. Такую разработанную систему нельзя будет назвать качественной, так как на этапе сопровождения могут появится проблемы, вызванные необходимостью переписывания исходный программных кодов (например, при изменении требований, предъявляемых к функциональности системы).

Для повышения качества исходного кода программ используется рефакторинг. В Википедии приводится следующее определение рефакторинга [9]: «Рефакторинг (refactoring) или реорганизация кода – процесс изменения

внутренней структуры программы, не затрагивающий её внешнего поведения и имеющий целью облегчить понимание её работы». Таким образом, для разработки высококачественного ПО необходимо, во-первых, реализовать требуемую функциональность системы, а во-вторых, обеспечить выполнение и нефункциональных требований, таких как удобство сопровождения. Под удобством сопровождения понимается простота внесения изменения в исходный код программы.

В источнике [10] приводятся следующие характеристики некачественно разработанного ПО:

1) жесткость - характеристика программы, затрудняющая внесение изменений в код;

2) хрупкость - свойство программы повреждаться во многих местах при внесении единственного изменения;

3) косность характеризуется тем, что код содержит части, которые могли бы оказаться полезными в других системах, но усилия и риски, сопряженные с попыткой отделить эти части кода от оригинальной системы, слишком велики;

4) ненужная сложность характеризуется тем, что программа содержит элементы, которые не используются в текущий момент;

5) ненужные повторения, которые состоят в повторяющихся фрагментах кода в программе;

6) непрозрачность, которая характеризует трудность кода для понимания.

Для написания качественного ПО при его разработке целесообразно использовать паттерны проектирования и применять следующие основные принципы при проектировании иерархии классов:

1) Принцип единственной обязанности (single responsibility principle) [11]. Этот принцип утверждает, что любой объект (экземпляр класса) должен иметь лишь одну обязанность, и она должна быть полностью инкапсулирована в класс. Следование данному принципу приводит к тому, что изменения, вносимые в класс, должны будут определяться только одной причиной, а не несколькими.

2) Принцип открытости/закрытости (open/closed principle) [12]. Данный принцип определяет, что программные сущности (классы, модули, функции) должны быть открыты для расширения, но закрыты для модификации. Следование данному принципу позволяет изменить поведение таких сущностей без изменения их исходного кода, что позволяет избежать лишних трудозатрат.

3) Принцип подстановки Лисков (Liskov substitution principle) [13]. Данный принцип, сформулированный Барбарой Лисков в 1987 г., определяет что должна существовать возможность вместо базового класса подставить любой его подтип. Следование данному принципу приводит к тому, что поведение классов-наследников становится предсказуемым для кода, использующего объекты их базовых классов.

4) Принцип разделения интерфейса (interface segregation principle) [14]. Данный принцип определяет, что слишком «толстые» интерфейсы необходимо разделять на более маленькие и специфические, чтобы клиенты маленьких интерфейсов знали только о методах, которые необходимы им в работе. Следование данному принципу приводит к тому, что при изменении метода интерфейса, не будут подлежать изменению клиенты, не использующие данный метод. Это в свою очередь позволяет оставаться системе гибкой при изменении логики ее работы и пригодной для рефакторинга.

5) Принцип инверсии зависимостей (dependency inversion principle) [15]. Данный принцип определяет, что, во-первых, модули верхних уровней не должны зависеть от модулей нижних уровней и оба типа модулей в свою очередь должны зависеть от абстракций, а во-вторых, абстракции не должны зависеть от деталей, а наоборот, детали должны зависеть от абстракций. Следование данному принципу позволяет уменьшить связанность в исходном программном коде.

При написании программ часто используются одни и те же схемы организации взаимодействия между объектами. За такими схемами взаимодействия закрепилось название паттерны или шаблоны проектирования. Использование паттернов позволяет создавать высококачественное ПО, которое можно легко модифицировать и сопровождать. При гибкой методологии разработки ПО целесообразно применять паттерны «Команда», «Стратегия», «Фасад», «Посредник», «Одиночка», «Фабрика», «Компоновщик», «Наблюдатель», «Абстрактный сервер/адаптер/шлюз», «Заместитель и Шлюз», «Посетитель и Состояние». С описанием этих и других паттернов можно ознакомится в источниках [16-18].

ЗАКЛЮЧЕНИЕ

Курс «Технология разработки программного обеспечения» является завершающим при обучении программированию студентов в магистратуре по направлению 230100.68 «Информатика и вычислительная техника». Этот курс можно считать продолжением изучения таких дисциплин, как «Информатика», «Программирование», «Объектно-ориентированное программирование», «Технологии программирования», осваиваемых в рамках бакалавриата. Однако дисциплины, посвященные различным аспектам программирования, при обучении в бакалавриате по направлению «Информатика и вычислительная техника», были ориентированы, прежде всего, на разработку ПО одним человеком.

В то же время, при реализации средних и крупных проектов, программисту практически всегда приходится работать в составе команды разработчиков. При работе в таких проектах отсутствие опыта коллективной разработки ПО будет снижать производительность труда программиста и оказывать негативное влияние на процесс его адаптации в команде разработчиков. Поэтому основное внимание при изучении дисциплины «Технология разработки ПО» уделяется методам командной разработки ПО.

Применение современных технологий разработки ПО позволяет сделать этот процесс более прогнозируемым, контролируемым, экономичным и управляемым. Естественно, что с течением времени происходит смена парадигм программирования, появляются новые языки и средства программирования, следовательно, претерпевает изменение и технология разработки ПО. Поэтому программисту очень важно научиться быстро изучать новые для него языки программирования, осваивать новые инструментальные средства и методы программирования. Для этого надо системно подходить к изучению нового для себя материала: выделить основные части, понять их логическую организацию и взаимосвязь, определить сходства и отличия от средств, изученных ранее.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Разработка программного обеспечения [Электронный ресурс] // Википедия: свободная энциклопедия. – Режим доступа: http://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F
2. Каскадная модель [Электронный ресурс] // Википедия: свободная энциклопедия. – Режим доступа: http://ru.wikipedia.org/wiki/%CA%E0%F1%EA%E0%E4%ED%E0%FF_%EC%EE%E4%E5%EB%FC
3. Boehm, B. A Spiral Model of Software Development and Enhancement/ B. Boehm // ACM SIGSOFT Software Engineering Notes. – 1986. – V. 11, I. 4. – P. 14-24.
4. V-модель [Электронный ресурс] // Википедия: свободная энциклопедия. – Режим доступа: http://ru.wikipedia.org/wiki/V-Model
5. Гибкая методология разработки [Электронный ресурс] // Википедия: свободная энциклопедия. – Режим доступа: http://ru.wikipedia.org/wiki/%D0%93%D0%B8%D0%B1%D0%BA%D0%B0%D1%8F_%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D1%8F_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8
6. Кент, Б. Экстремальное программирование /Б. Кент. - СПб.: Питер, 2002. -212 с.
7. Кент, Б. Экстремальное программирование: разработка через тестирование/ Б. Кент. – СПб.: Питер, 2003. – 224 с.
8. Microsoft Solution Framework [Электронный ресурс] // Википедия: свободная энциклопедия. – Режим доступа: http://ru.wikipedia.org/wiki/Microsoft_Solutions_Framework
9. Рефакторинг [Электронный ресурс] // Википедия: свободная энциклопедия. – Режим доступа: http://ru.wikipedia.org/wiki/%D0%E5%F4%E0%EA%F2%EE%F0%E8%ED%E3
10. Лекция 12. Обеспечение качества программных продуктов [Электронный ресурс] // ИНТУИТ. Национальный открытый университет. – Режим доступа: http://www.intuit.ru/studies/courses/4806/1054/lecture/16133
11. Принцип единственной обязанности [Электронный ресурс] // Википедия: свободная энциклопедия. – Режим доступа: http://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BD%D1%86%D0%B8%D0%BF_%D0%B5%D0%B4%D0%B8%D0%BD%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D0%BE%D0%B9_%D0%BE%D0%B1%D1%8F%D0%B7%D0%B0%D0%BD%D0%BD%D0%BE%

D1%81%D1%82%D0%B8
12. Принцип открытости/закрытости [Электронный ресурс] // Википедия: свободная энциклопедия. – Режим доступа: http://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BD%D1%86%D0%B8%D0%BF_%D0%BE%D1%82%D0%BA%D1%80%D1%8B%D1%82%D0%BE%D1%81%D1%82%D0%B8/%D0%B7%D0%B0%D0%BA%D1%80%D1%8B%D1%82%D0%BE%D1%81%D1%82%D0%B8
13. Принцип подстановки Барбары Лисков [Электронный ресурс] // Википедия: свободная энциклопедия. – Режим доступа: http://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BD%D1%86%D0%B8%D0%BF_%D0%BF%D0%BE%D0%B4%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BA%D0%B8_%D0%91%D0%B0%D1%80%D0%B1%D0%B0%D1%80%D1%8B_%D0%9B%D0%B8%D1%81%D0%BA%D0%BE%D0%B2
14. Принцип разделения интерфейса [Электронный ресурс] // Википедия: свободная энциклопедия. – Режим доступа: http://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BD%D1%86%D0%B8%D0%BF_%D1%80%D0%B0%D0%B7%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81%D0%B0
15. Принцип инверсии зависимостей [Электронный ресурс] // Википедия: свободная энциклопедия. – Режим доступа: http://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BD%D1%86%D0%B8%D0%BF_%D0%B8%D0%BD%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D0%B8_%D0%B7%D0%B0%D0%B2%D0%B8%D1%81%D0%B8%D0%BC%D0%BE%D1%81%D1%82%D0%B5%D0%B9
16. Паттерны проектирования/ Эрик Фримен, Элизабет Фримен, К. Сьерра, Б. Бейтс. – СПб: Питер, 2011. – 656 с.
17. Смит, Дж. Элементарные шаблоны проектирования/ Дж. Смит. – М.: Вильямс, 2013. – 304 с.
18. Нильсон, Дж. Применение DDD и шаблонов проектирования: проблемно-ориентированное проектирование приложений с примерами на C# и .NET / Дж. Нильсон. – М.: Вильямс, 2008. – 560 с.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 3
1. ОСНОВЫ ПРОЦЕССА РАЗРАБОТКИ ПО 4
2. ОСНОВНЫЕ ПАРАДИГМЫ ПРОЦЕССА РАЗРАБОТКИ ПО 8
2.1. Каскадная модель разработки ПО 8
2.2. Итеративная модель разработки ПО 9
2.3. Спиральная модель разработки ПО 11
2.4. V модель разработки ПО 13
3. МЕТОДОЛОГИИ ПРОГРАММИРОВАНИЯ 16
3.1. Agile методологии программирования 16
3.2. Методология TDD 28
3.3. Методология MSF 36
4. ТЕХНОЛОГИЯ РАЗРАБОТКИ ПО С ИСПОЛЬЗОВАНИЕМ ИНСТРУМЕНТАЛЬНЫХ СРЕДСТВ MICROSOFT 47
4.1. Сервер Team Foundation Server (TFS) 47
4.2. Организация командной разработки на базе Visual Studio и TFS 54
4.3. Обеспечение качества разрабатываемого ПО 67
ЗАКЛЮЧЕНИЕ 72
БИБЛИОГРАФИЧЕСКИЙ СПИСОК 73

Учебное издание

Андрей Валерьевич Машкин

ТЕХНОЛОГИЯ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Учебное пособие

Редактор Н.В. Сажина

Подписано в печать 06.03.2014 г.
Формат 60x90/16. Бумага офисная.
Печать офсетная. Усл. печ. л. .
Тираж экз. Заказ .

Отпечатано: РИО ВоГУ
160000, г. Вологда, ул. Ленина, 15.