Свиридова О.В., Рыбанов А.А., Макушкина Л.А.

Технология командной разработки программных систем

Волжский

2019

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ВОЛЖСКИЙ ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ (ФИЛИАЛ)
ФЕДЕРАЛЬНОГО ГОСУДАРСТВЕННОГО БЮДЖЕТНОГО ОБРАЗОВАТЕЛЬНОГО
УЧРЕЖДЕНИЯ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОЛГОГРАДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Свиридова О.В., Рыбанов А.А., Макушкина Л.А.

Технология командной разработки программных систем

Электронное учебно-методическое пособие

2019

УДК 004.45(07)
ББК 32.81я73
С 247

Р е ц е н з е н т ы:
доцент кафедры математики и информатики ФГБОУ ВПО «Волгоградский государственный медицинский университет»,
кандидат социологических наук
Плешакова Е.О.,
заведующий кафедрой информационных систем, математики и методик обучения Приамурского государственного университета им. Шолом-Алейхема, канд. пед. наук, доцент
Баженов Р.И.

Издается по решению редакционно-издательского совета
Волгоградского государственного технического университета

Свиридова, О.В.
Технология командной разработки программных систем [электронный ресурс] : учебно-методическое пособие / О.В. Свиридова, А.А. Рыбанов, Л.А. Макушкина ; ВПИ (филиал) ВолгГТУ, – Электрон. текстовые дан. (1 файл: 673 КБ). – Волжский, 2019. – Режим доступа: http://lib.volpi.ru. – Загл. с титул. экрана.
ISBN 978-5-9948-3139-7

В учебно-методическом пособии рассматриваются подходы к организации командной разработки программных приложений. Основное внимание уделяется методологии и решениям Microsoft в части управления жизненным циклом программных приложений: Visual Studio, Team Foundation Server. В курсе лекций рассматриваются современные технологии разработки программного обеспечения, процессы командной разработки ПО, анализируются формальные и гибкие технологии разработки ПО.

Предназначено для студентов, обучающихся по направлению 09.03.04 «Программная инженерия» в рамках курса «Технология командной разработки программных систем».

Ил. 12, табл. 4, библиограф.: 7 назв.

ISBN 978-5-9948-3139-7

Содержание

ВВЕДЕНИЕ

Современное состояние бизнеса требует от создателей программного обеспечения (*ПО*) разработки программных продуктов высокого качества в рамках отведенного бюджета и в срок. В создании программных продуктов, как правило, принимают участие различные специалисты, которые объединяются в команды. Команды могут включать сотрудников организации разработчика и заказчика, привлекаемых временных специалистов и субподрядчиков. Члены команды разработчиков *ПО* могут территориально находиться в одном или разных местах (распределенная разработка). Эффективное решение задач создания качественного *ПО* предполагает использование инструментальных средств, методик и технологий управления процессами жизненного *цикла* программных систем: формирования требований, моделирования и проектирования, разработки, тестирования, построения и развертывания систем.

Рациональная организация процессов разработки программных систем описывается в стандартах (международных, государственных, корпоративных), которые часто называют методологиями разработки *ПО*. Методологии создания *ПО* обычно разрабатываются ведущими производителями программных систем и их сообществами с учетом особенностей программных продуктов, а также сферы внедрения. Методологии описывают подходы к организации рациональной стратегии и возможному набору процессов создания *ПО*.

В настоящее время все большее распространение получают гибкие методологии разработки программного обеспечения, где основное внимание сосредоточено на создании качественного продукта, а не подготовку исчерпывающей документации по проекту. При этом акцент делается на организацию эффективного управления командой. Как отмечает Эрих Гамма: "…*ключ* к своевременной поставке продукта – не процессы, а лю-

ди"[1]. *Самоорганизация* и целеустремленность команды разработчиков позволяет создавать высококачественные программные продукты в сжатые сроки.

Инструменты управления жизненным циклом приложений во многом способствуют успешности программных проектов. Компания Microsoft предоставляет разработчикам гибкий *инструментарий* для управления жизненным циклом приложений – ALMVisualStudio и TeamFoundationServer. Традиционные *средства разработки* программ в VisualStudio дополнены средствами архитектурного проектирования и тестирования. *Инструментарий* TeamFoundationServer позволяет формировать и отслеживать требования к программной системе, связывать их с задачами и реализацией, распределять между членами команды, проводить построение программного продукта, управлять тестированием, проводить *контроль версий*, предоставлять средства коммуникации с членами команды и заказчиками, подготавливать многочисленные отчеты.

Данный курс имеет цель представить основные положения командной разработки программного обеспечения, управления жизненным циклом приложений, гибкой методологии создания программных систем, а также возможностей инструментария VisualStudio2012 и TeamFoundationServe для управления жизненным циклом приложений.

1. Введение в технологии разработки программного обеспечения

Введение

В соответствии с определением, данным в Википедии [2], *технология* – комплекс организационных мер, операций и приемов, направленных на изготовление, обслуживание, ремонт и/или эксплуатацию изделия с номинальным качеством и оптимальными затратами, и обусловленных текущим уровнем развития науки, техники и общества в целом. *Технология разработки программного обеспечения* (*ПО*) представляет собой комплекс организационных мер, операций и приемов, направленных на разработку программных продуктов высокого качества в рамках отведенного бюджета и в срок. Технологии включают методики, методологии, средства и процедуры разработки *ПО*.

Модели жизненного цикла программного обеспечения

В настоящее время существует достаточно много различных методик разработки программного обеспечения [3, 4]. Методики различаются используемой моделью жизненного *цикла ПО* и уровнем формализма при его создании. *Жизненный цикл программного обеспечения* (*ПО*) – период времени, который начинается с момента принятия решения о необходимости создания программного продукта и заканчивается в момент его полного изъятия из эксплуатации.

Классический *жизненный цикл* определяется *каскадной* или *водопадной моделями*, в которых предполагается последовательное выполнение этапов создания *ПО*: *анализ*, проектирование, разработка, тестирование и сопровождение.

Методики, базирующиеся на каскадной модели, характеризуются тем, что переход на следующую стадию проектирования осуществляется только после того, как будет завершена работа на текущей стадии. Возврат на предыдущие стадии не предполагается. Изменения, вносимые в проект, могут сильно повлиять на время и сроки проектирования. Обычные сроки

реализации проекта 6 - 12 месяцев. Такие методики применимы к простым программным системам, когда неопределенность в проекте минимальна и изменений в процессе проектирования не предполагается, а используемые технологические решения проверены и хорошо известны членам команды.

Развитием и усовершенствованием каскадной *модели жизненного цикла ПО* является *итерационная спиральная модель*, в которой разработка *ПО* осуществляется по спирали. Каждый виток (*итерация*) спирали предполагает реализацию определенного функционала программной системы. На каждом витке разработки реализуются такие же этапы создания *ПО*, как и в каскадной модели, то есть: *анализ*, проектирование, разработка и тестирование. Количество витков в спиральной модели не регламентировано и определяется разработчиком при выделении приоритетов пользовательских или функциональных требований к программной системе. Средняя продолжительность проектов 6 - 12 месяцев, а продолжительность итерации: 3 - 6 месяцев.

Спиральная модель жизненного *цикла ПО* лежит в основе методологии создания ИТ-решений компании Microsoft – *MSF* (MicrosoftSolutionFramework). В данной методологии компания Microsoft отразила свое видение на процессы создания программных систем различного назначения.

В *инкрементной итерационной модели жизненного цикла ПО* разработка реализуется несколькими итерациями с постепенным наращиванием функциональности системы. Каждая *итерация* предполагает планирование, *анализ* риска, разработку и оценку результатов итерации.

К итеративным методам разработки *ПО* относится методология, созданная компанией RationalSoftware – RationalUnifiedProcess (*RUP*). *Унифицированный процесс RUP* определяет виды деятельности, необходимые для проектирования программного продукта на основе требований пользователя, которые могут изменяться в процессе разработки си-

стемы. Данный процесс может быть адаптирован для разработки различных программных систем.

По степени формализма методологии отличаются количеством используемых процессов и создаваемых артефактов (документов, отчетов), а также формальности процедур рецензирования.

Методики призваны обеспечить целенаправленный *процесс управления* действиями всех заинтересованных лиц в проекте создания программного обеспечения: заказчиков, пользователей, разработчиков и руководство. Для эффективного управления процессом разработки программных систем необходимо сформировать подход, который обеспечивал:

- управление и мониторинг деятельности команды проекта;
- распределение работ между участниками проекта;
- формирование перечня требований к разрабатываемым элементам проекта и документации;
- определение набора критериев качества программного продукта.

Существующие методологии и методики разработки *ПО* предполагают выполнение определенных процессов на всех этапах жизненного *цикла* программной системы. Процессы создания *ПО* должны соответствовать определенным требованиям для обеспечения качества выходного продукта.

Зрелость процессов разработки ПО

Американским университетом Карнеги-Меллон (SoftwareEngineeringInstitute, *SEI*) разработана модель *CMMI* (CapabilityMaturityModelIntegration), характеризующая *уровни зрелости* процесса разработки *ПО* [3]. Модель *CMMI* представляет описание идеального процесса разработки *ПО*. Базовым понятием модели CMMI считается зрелость компании.

Незрелой называют компанию, где процесс создания *ПО* и принимаемые решения зависят только от таланта конкретных разработчиков. Ре-

зультатом является высокий риск превышения бюджета или срыва сроков окончания проекта.

В зрелой компании работают ясные процедуры управления проектами и построения программных продуктов. По мере необходимости эти процедуры уточняются и развиваются. Оценки длительности и затрат разработки точны, основываются на накопленном опыте. Кроме того, в компании имеются и действуют корпоративные стандарты на процессы взаимодействия с заказчиком, процессы анализа, проектирования, программирования, тестирования и внедрения программных продуктов. Все это создает среду, обеспечивающую качественную разработку программного обеспечения.

Следует отметить, что методологии разработки *ПО*, базирующиеся на модели *CMMI*, унифицированном процессе *RUP* и модели *MSF* компании Microsoft являются достаточно сильно формализованными, предполагают ограничения по организационной структуре, бюрократический подход ко многим проектным процедурам, наличие большого количества сопроводительных документов, сложные процедуры согласования изменений проекта и принятия других проектных решений. Такие методологии разработки *ПО* оправдывают себя для крупных проектов с большим количеством исполнителей, заказчиками которых являются государственные или бюрократические организации.

Для небольших команд (до 10 участников) альтернативой строго формализованных подходов к разработке *ПО* являются гибкие (agile) методологии. Гибкие методологии ориентированы на профессионалов, которые мотивированы на создание качественного программного продукта в кратчайшие сроки. Основными положениями гибкого подхода к созданию *ПО* являются [6]:

- люди и взаимодействие важнее процессов и программных средств;
- работающее ПО важнее исчерпывающей документации;

• взаимодействие с заказчиком важнее согласования условий контакта;

• готовность к изменениям важнее следования первоначальному плану.

Гибкие методологии ориентированы на минимизацию рисков, реализуя короткие итерации длительностью в одну или две недели. Каждая *итерация* заканчивается выпуском заданной функциональности программного проекта и реализует этапы *работ* по планированию, анализу требований, проектированию, кодированию, тестированию и *документированию*. Отдельная *итерация*, как правило, недостаточна для выпуска новой версии продукта, но подразумевается, что гибкий программный проект готов к выпуску в конце каждой итерации. По окончании каждой итерации *команда* выполняет переоценку приоритетов разработки.

Основной целью использования тех или иных методологий является создание предсказуемого процесса разработки *ПО* заданного качества. При этом, дополнительные *затраты*, связанные с поддержкой методологии (создание различных артефактов, отчетов, выполнение заданных процедур), должны быть достаточными и минимально необходимыми для обеспечения требуемой эффективности проектов.

ИТ-решения по управлению жизненным циклом ПО

Улучшению процессов создания программного обеспечения служит методология *управления жизненным циклом приложений* (*ALM* - applicationlifecyclemanagement), которая представляет собой концепцию управления программным проектом на всех этапах его жизни. *ALM* определяет непрерывный *процесс управления* жизненным циклом приложения по его управлению, развитию и обслуживанию. Принципы *ALM* реализуются ИТ-решениями различных вендоров.

Решение HP *ALM* onSaaS компании Hewlett-Packard способствует ускорению процессов консолидации; в его рамках доступны услуги коман-

ды экспертов по платформе HP *ALM*, имеется упрощенная система управления, встроенная возможность осуществлять масштабирование по требованию, а также предоставляется *поддержка*, необходимая для того, чтобы сосредоточиться на инновациях.

С помощью решений *Open ALM* от компании Borland организации смогут эффективно использовать свои существующие ресурсы и средства для разработки. Это поможет достигнуть прозрачности, контроля и дисциплинированности на всем протяжении *цикла* создания *ПО*.

Компания *IBM* для управления жизненным циклом приложений предлагает решение *IBM*® Rational® ClearQuest®. ИТ-решение поддерживает рационализированный, динамичный процесс разработки приложений, который одновременно является ориентированным на роли и управляемым процессами. Проекты определяют *контекст* выполнения заданий; их *безопасность* можно обеспечить через *определение* политик безопасности и ролей. Работа может быть распределена между членами коллектива, которые находятся в одном месте или в разных местах. Кроме того, работа является трассируемой до исходного запроса и до проекта, который реализуется *по* этому запросу.

Компания Microsoft предлагает набор средств в *Visual Studio* 2012 и объединения этих средств с *Visual Studio Team FoundationServer* для управления жизненным циклом приложений. В основе решений Microsoft по управлению жизненным циклом приложений лежат следующие принципы: продуктивность, *интеграция* и *расширяемость*. Продуктивность достигается обеспечением командной работы над проектом и управлением сложностью. *Интеграция* реализуется наличием полнофункциональных возможностей в единой среде проектирования, разработки, тестировании и сопровождении программного приложения, а также прозрачностью процесса создания *ПО* для всех участников проекта. *Расширяемость* поддерживается интегрированной средой разработки (*IDE*) и от-

крытостью платформы для расширения и создания собственных инструментов, которые интегрируются с *Team Foundation Server*.

Вопросы

1. Что включает понятие "технология разработки программного обеспечения"?
2. Что определяет жизненные цикл программного обеспечения?
3. Поясните содержание каскадной модели разработки программного обеспечения.
4. Поясните содержание итерационной спиральной модели разработки программного обеспечения.
5. Поясните содержание итеративной модели разработки программного обеспечения.
6. Что должен обеспечивать эффективный подход к управлению процессом разработки ПО?
7. Что понимается под зрелостью процессов для компании, разрабатывающей ПО?
8. Приведите основные положения гибкого подхода к созданию ПО.
9. Приведите основное назначение методологии управления жизненным циклом приложений.
10. Какие инструментальные средства предлагает компания Microsoft для управления жизненным циклом приложений?

2. Процессы командной разработки программного обеспечения MSF

Введение

Microsoft Solutions Framework (*MSF*) является методологией разработки *ПО*, которая представляет собой *обобщение* лучших проектных практик, которые использовались командами разработчиков Microsoft. Данная методология описывает управление людьми и рабочими процессами при разработке ИТ-решений.

ИТ-решение - понимается как скоординированная поставка набора элементов (таких как *программные средства*, документация, обучение и сопровождение), необходимых для удовлетворения бизнес-потребности конкретного заказчика.

Концепция управления жизненным циклом приложений, принятая разработчиками *ПО*, привела к тому, что методология *MSF* стала составной часть продукта *Visual Studio Team System* (VSTS), который реализовывал подход Microsoft в плане – *ALM*. В продукт VSTS вошли шаблоны процессов для реализации положений модели *CMMI* - *MSF* for *CMMI* и моделей гибкой разработки *ПО* – *MSF* for Agileи Scrum. Таким образом, VSTS являются инструментарием управления жизненным циклом приложений, который позволяет создавать программные системы различного назначения в командах, придерживающихся различных подходов к управлению процессами разработки *ПО*.

Основными являются следующие принципы *MSF*.

1. *Единое видение проекта*, которое предполагает понимание всеми заинтересованными лицами целей и задач создания ПО.

2. *Гибкость – готовность к переменам*, что обеспечивает возможность уточнения и изменения требований в процессе разработки ПО,

оперативного и быстрого реагирования на текущие изменения условий проекта при неизменной эффективности управленческой деятельности.

3. *Концентрация на бизнес-приоритетах*, что предполагает создание продукта с высоким потребительским качеством и формирование определенной выгоды или отдачи. Для организаций, как правило, это получение прибыли.

4. *Поощрение свободного общения*, что предполагает открытый и честный обмен информацией как внутри команды, так и с ключевыми заинтересованными лицами.

Универсальность модели *MSF* определяется тем, что благодаря своей гибкости и отсутствию жестко установленных связей и процедур она может быть применена при разработке различных программных приложений, которые могут использоваться в бизнесе и повседневной жизни.

Модель *MSF* базируется на сочетании двух моделей жизненного *цикла* программных систем: каскадной и спиральной (см. рис. 2.1).

Рис. 2.1. Модель жизненного цикла решения MSF

В основе методологии *MSF* лежит *итеративный интегрированный подход* к созданию и внедрению решений, базирующийся на фазах и вехах.

Итеративность подхода предусматривает поэтапное создание работоспособной программной системы с определенной функциональностью, отражающей требования к конечному продукту на данном этапе разработки. Каждый виток спирали состоит из идентичных фаз, на которых выполняются этапы *работ по* формированию концепции фазы, планированию, разработке, стабилизации и внедрению. Набор требований программной системы и соответствующих им задач, которые реализуются на заданном витке спирали, определяется менеджером проекта на основе ранжирования требований к системе. Каждый последующий виток спирали добавляет к программной системе функциональность, отражающую *требования заказчика*.

Интеграция в рамках одного проекта процедур разработки и внедрения системы позволяет представлять заказчику различные промежуточные работоспособные версии программного продукта, оперативно вносить изменения, отражающие видение заказчиком функциональности и дизайна проектируемой системы, снизить риски проекта за счет раннего выявления проблем и возможности своевременного их устранения.

Фазы проекта определяют последовательно решаемые задачи, а вехи (milestones) – ключевые точки проекта, характеризующие достижение какого-либо существенного результата.

В *MSF* используются два вида вех: главные и промежуточные. Они имеют следующие характеристики:

- главные вехи служат точками перехода от одной фазы к другой и определяют изменения в текущих задачах ролевых кластеров проектной команды; в MSF главные вехи являются в достаточной степени универсальными для применения в любом ИТ проекте;

- промежуточные вехи показывают достижение определенного прогресса в исполнении фазы проекта и расчленяют большие сегменты работы на меньшие, обозримые и управляемые участки; промежуточные вехи могут варьироваться в зависимости от характера проекта.

Модель команд

Главной особенностью модели команды в *MSF* является то, что она не имеет официального лидера. Все отвечают за проект в равной степени, уровень заинтересованности каждого в результате очень высок, а коммуникации внутри группы четкие, ясные, дружественные и ответственные.

Такое возможно при высоком уровне самосознания и заинтересованности каждого члена команды, а также при достаточно высоком уровне профессионализма.

Одной из особенностей отношений внутри команды является высокая культура дисциплины обязательств:

- готовность работников принимать на себя обязательства перед другими;
- четкое определение тех обязательств, которые они на себя берут;
- стремление прилагать должные усилия к выполнению своих обязательств;
- готовность честно и незамедлительно информировать об угрозах выполнению своих обязательств.

Ролевые кластеры *MSF* основаны на семи качественных целях, достижение которых определяет успешность проекта. Эти цели обуславливают *модель проектной группы* и образуют ролевые кластеры (или просто роли) в проекте. Каждый *кластер* может включать одного или нескольких специалистов. Каждый *ролевой кластер* представляет уникальную точку зрения на проект, и в то же время никто из членов проектной группы в одиночку не в состоянии успешно представлять все возможные взгляды, отражающие качественно различные цели.

В *MSF* следующие ролевые кластеры (таблица 2.1):

Таблица 2.1. Ролевые кластеры

Управление продуктом	Основная задача кластера – обеспечить, чтобы заказчик остался довольным в результате выполнения проекта. Этот ролевой кластер в проекте представляет интересы заказчика. Он представляет бизнес-сторону проекта и обеспечивает его согласованность со стратегическими целями заказчика.
Управление программой	Кластер обеспечивает управленческие функции – отслеживание планов и их выполнение, ответственность за бюджет, ресурсы проекта, разрешение проблем и трудностей процесса, создание условий, при которых команда может работать эффективно, испытывая минимум бюрократических преград.
Разработка	Кластер обеспечивает разработку кода приложения.
Тестирование	Кластер отвечает за тестирование ПО.
Удовлетворение потребителя	Кластер решает задачи пользовательского дизайна приложения и обеспечения удобства эксплуатации ПО, обучение пользователей работе с ПО, создание пользовательской документации.
Управление выпуском	Кластер отвечает за внедрение проекта и его функционирование, берет на себя связь между разработкой решения, его внедрением и последующим сопровождением, обеспечивая информированность членов проектной группы о последствиях их решений.
Архитектура	Кластер отвечает за организацию и выполнение высокоуровневого проектирования решения, создание функциональной спецификации ПО и управление этой спецификацией в процессе разработки, определение рамок проекта и ключевых компромиссных решений.

Изменения в задачах ролевых кластеров проектной команды происходят по мере смены фаз проекта. Переход от одной фазы к другой включает в себя также перенос основной ответственности от одних ролевых кластеров к другим, как показано в таблице 2.2.

Таблица 2.2. Распределение ответственности ролевых кластеров

Веха	Ведущие ролевые кластеры
Концепция утверждена	Управление продуктом
Планы проекта утверждены	Управление программой
Разработка завершена	Разработка, удовлетворение потребителя
Готовность решения утверждена	Тестирование, управление выпуском
Внедрение завершено	Управление выпуском

Масштабирование команды MSF

В зависимости от размера и сложности проекта модель команд *MSF* допускает масштабирование. Для небольших и несложных проектов один сотрудник может объединять несколько ролей. При этом некоторые роли нельзя объединять. В таблице 2.3 представлены рекомендации *MSF* относительно совмещения ролей в рамках проекта одним членом команды. "+" означает, что совмещение возможно, "+-" - что совмещение возможно, но нежелательно, "-" означает, что совмещение не рекомендуется.

Таблица 2.3. Рекомендации MSF относительно совмещения ролей в команде проекта

	Управление продуктом	Управление программой	Разработка	Тестирование	Удовлетворение потребителя	Управление выпуском	Архитектура
Управление продуктом		-	-	+	+	+-	-
Управление программой	-		-	+-	+-	+	+
Разработка	-	-		-	-	-	+
Тестирование	+	+-	-		+	+	+-
Удовлетворение потребителя	+	+-	-	+		+-	+-
Управление выпуском	+-	+	-	+	+-		+
Архитектура	-	+	+	+-	+-	+	

В частности, нельзя совмещать разработку и тестирование, поскольку необходимо, чтобы у тестировщиков был сформирован свой, независимый взгляд на систему, базирующийся на изучении требований.

Для больших команд (более 10 человек) *модель проектной группы MSF* предлагает *разбиение* на малые многопрофильные *группы направлений*. Эти малые коллективы работают параллельно, регулярно синхронизируя свои усилия, каждый из которых устроен на основе модели кластеров. Такие команды имеют четко определенную задачу и ответственны за все относящиеся к ней вопросы, начиная от проектирования и составления календарного *графика*.

Кроме того, когда ролевому кластеру требуется много ресурсов, формируются так называемые *функциональные группы*, которые затем объединяются в ролевые кластеры. Они создаются в больших проектах, когда необходимо сгруппировать работников внутри ролевых кластеров по областям их компетенций. Часто функциональные группы имеют внутреннюю иерархическую структуру. Например, менеджеры программы могут быть подотчетны ведущим менеджерам программы, которые, в свою очередь, отчитываются перед главным менеджером программы. Подобные структуры могут также появляться внутри областей компетенций.

Управление компромиссами. Хорошо известна взаимозависимость между ресурсами проекта (людскими и финансовыми), его календарным графиком (временем) и реализуемыми возможностями (функциональность). Эти три переменные образуют треугольник, показанный на рис. 2.2.

Рис. 2.2. Треугольник компромиссов

После достижения равновесия в этом треугольнике изменение на любой из его сторон для поддержания баланса требует модификаций на другой (двух других) сторонах и/или на изначально измененной стороне.

В проектной практике может фиксироваться один из ресурсов, тогда в процессе проектирования системы измерению подлежат только два оставшихся ресурса. Данная ситуация задается матрицей компромиссов, которая приведена на рис. 2.3. Так, если фиксируются ресурсы, то *время выполнения* проекта согласовывается, а функциональные возможности подлежат изменению.

	Согласовывается	Фиксируется	Принимается
Ресурсы	*		
Время		*	
Функции			*

Рис. 2.3. Матрица компромиссов

Вопросы

1. Как определяет понятие "ИТ-решение" компания Microsoft?
2. Назовите основные принципы MSF.
3. Чем определяется универсальность модели MSF?

4. Какая модель цикла программной системы используется в MSF?

5. В чем состоит итеративность методологии MSF?

6. Поясните назначение интеграции в методологии MSF?

7. В чем проявляется высокая культура дисциплины обязательств методологии MSF?

8. Назовите ролевые кластеры модели команд методологии MSF.

9. Как можно масштабировать команду, использующую методологию MSF?

10. Поясните назначение треугольника компромиссов.

3. Гибкие технологии разработки ПО

Введение

Гибкая методология разработки программного обеспечения ориентирована на использование итеративного подхода, при котором *программный продукт* создается постепенно, небольшими шагами, включающими реализацию определенного набора требований. При этом предполагается, что требования могут изменяться. Команды, использующие гибкие методологии, формируются из универсальных разработчиков, которые выполняют различные задачи в процессе создания программного продукта.

При использовании гибких методологий *минимизация* рисков осуществляется путём сведения разработки к серии коротких циклов, называемых *итерациями*, продолжительностью 2-3 недели. *Итерация* представляет собой набор задач, запланированных на выполнение в определенный период времени. В каждой итерации создается работоспособный вариант программной системы, в которой реализуются наиболее приоритетные

(для данной итерации) *требования заказчика*. На каждой итерации выполняются все задачи, необходимые для создания работоспособного программного обеспечения: планирование, *анализ* требований, проектирование, *кодирование*, тестирование и *документирование*. Хотя отдельная *итерация*, как правило, недостаточна для выпуска новой версии продукта, подразумевается, что текущий *программный продукт* готов к выпуску в конце каждой итерации. По окончании каждой итерации *команда* выполняет переоценку приоритетов требований к программному продукту, возможно, вносит коррективы в разработку системы.

Принципы и значение гибкой разработки

Для методологии гибкой разработки декларированы ключевые постулаты, позволяющие командам достигать высокой производительности:

- люди и их взаимодействие;
- доставка работающего программного обеспечения;
- сотрудничество с заказчиком;
- реакция на изменение.

Люди и взаимодействие. Люди – важнейшая составная часть успеха. Отдельные члены команды и хорошие коммуникации важны для высокопроизводительных команд. Для содействия коммуникации гибкие методы предполагают частые обсуждения результатов работы и внесение изменений в решения. Обсуждения могут проводиться ежедневно длительностью несколько минут и по завершению каждой итерации с анализом результатов работ и ретроспективой. Для эффективных коммуникаций при проведении собраний участники команд должны придерживаться следующих ключевых правил поведения:

- уважение мнения каждого участника команды;
- быть правдивым при любом общении;
- прозрачность всех данных, действий и решений;
- уверенность, что каждый участник поддержит команду;

- приверженность команде и ее целям.

Для создания высокопроизводительных команд в гибких методологиях кроме эффективной команды и хороших коммуникаций необходим совершенный программный *инструментарий*.

Работающее программное обеспечение важнее всеобъемлющей документации. Все гибкие методологии выделяют необходимость доставки заказчику небольших фрагментов работающего программного обеспечения через заданные интервалы. *Программное обеспечение*, как правило, должно пройти уровень модульного тестирования, тестирования на уровне системы. При этом объем документации должен быть минимальным. В процессе проектирования *команда* должна поддерживать в актуальном состоянии короткий документ, содержащий обоснования решения и описание структуры.

Сотрудничество с заказчиком важнее формальных договоренностей по контракту. Чтобы проект успешно завершился, необходимо регулярное и частое общение с заказчиком. Заказчик должен регулярно участвовать в обсуждении принимаемых решений по программному обеспечению, высказывать свои пожелания и замечания. Вовлечение заказчика в процесс разработки программного обеспечения необходимо создания качественного продукта.

Оперативное реагирование на изменения важнее следования плану. Способность реагирования на изменения во многом определяет успех программного проекта. В процессе создания программного продукта очень часто изменяются *требования заказчика*. Заказчики очень часто точно не знают, чего хотят, до тех пор, пока не увидят работающее *программное обеспечение*. Гибкие методологии ищут обратную связь от заказчиков в процессе создания программного продукта. Оперативное реагирование на изменения необходимо для создания продукта, который удовлетворит заказчика и обеспечит ценность для бизнеса.

Постулаты гибкой разработки поддерживаются 12 принципами [11]. В конкретных методологиях гибкой разработки определены процессы и правила, которые, в большей или меньшей степени, соответствуют этим принципам. Гибкие методологии создания программных продуктов основываются на следующих принципах [12]:

1. Высшим приоритетом считать удовлетворение пожеланий заказчика посредством поставки полезного программного обеспечения в сжатые сроки с последующим непрерывным обновлением. Гибкие методики подразумевают быструю поставку начальной версии и частые обновления. Целью команды является поставка работоспособной версии в течение нескольких недель с момента начала проекта. В дальнейшем программные системы с постепенно расширяющейся функциональностью должны поставляться каждые несколько недель. Заказчик может начать промышленную эксплуатацию системы, если посчитает, что она достаточно функциональна. Также заказчик может просто ознакомиться с текущей версией программного обеспечения, предоставить свой отзыв с замечаниями.

2. Не игнорировать изменение требований, пусть даже на поздних этапах разработки. Гибкие процессы позволяют учитывать изменения для обеспечения конкурентных преимуществ заказчика. Команды, использующие гибкие методики, стремятся сделать структуру программы качественной, с минимальным влиянием изменений на систему в целом.

3. Поставлять новые работающие версии ПО часто, с интервалом от одной недели до двух месяцев, отдавая предпочтение меньшим срокам. При этом ставится цель поставить программу, удовлетворяющую потребностям пользователя, с минимумом сопроводительной документации.

4. Заказчики и разработчики должны работать совместно на протяжении всего проекта. Считается, что для успешного проекта заказчики, разработчики и все заинтересованные лица должны общаться часто и помногу для целенаправленного совершенствования программного продукта.

5. Проекты должны воплощать в жизнь целеустремленные люди. Создайте команде проекта нормальные условия работы, обеспечьте необходимую поддержку и верьте, что члены команды доведут дело до конца.

6. Самый эффективный и продуктивный метод передачи информации команде разработчиков и обмена мнениями внутри неё – разговор лицом к лицу. В гибких проектах основной способ коммуникации – простое человеческое общение. Письменные документы создаются и обновляются постепенно по мере разработки ПО и только в случае необходимости.

7. Работающая программа – основной показатель прогресса в проекте. О приближении гибкого проекта к завершению судят по тому, насколько имеющаяся в данный момент программа отвечает требованиям заказчика.

8. Гибкие процессы способствуют долгосрочной разработке. Заказчики, разработчики и пользователи должны быть в состоянии поддерживать неизменный темп сколь угодно долго.

9. Непрестанное внимание к техническому совершенству и качественному проектированию повышает отдачу от гибких технологий. Члены гибкой команды стремятся создавать качественный код, регулярно проводя рефакторинг.

10. Простота – искусство достигать большего, делая меньше. Члены команды решают текущие задачи максимально просто и качественно. Если в будущем возникнет какая-либо проблема, то в качественный код имеется возможность внести изменения без больших затрат.

11. Самые лучшие архитектуры, требования и проекты выдают самоорганизующиеся команды. В гибких командах задачи поручаются не отдельным членам, а команде в целом. Команда сама решает, как лучше всего реализовать требования заказчика. Члены команды совместно работают над всеми аспектами проекта. Каждому участнику разрешено вносить свой

вклад в общее дело. Нет такого члена команды, который единолично отвечал бы за архитектуру, требования или тесты.

12. Команда должна регулярно задумываться над тем, как стать ещё более эффективной, а затем соответственно корректировать и подстраивать свое поведение. Гибкая команда постоянно корректирует свою организацию, правила, соглашения и взаимоотношения.

Вышеприведенным принципам, в определенной степени, соответствуют ряд методологий разработки программного обеспечения:

AgileModeling — набор понятий, принципов и приёмов (практик), позволяющих быстро и просто выполнять моделирование и документирование в проектах разработки программного обеспечения;

AgileUnifiedProcess(AUP) — упрощенная версия IBM RationalUnifiedProcess (RUP), которая описывает простое и понятное приближение (модель) для создания программного обеспечения для бизнес-приложений;

OpenUP — это итеративно-инкрементальный метод разработки программного обеспечения. Позиционируется как лёгкий и гибкий вариант RUP;

AgileDataMethod — группа итеративных методов разработки программного обеспечения, в которых требования и решения достигаются в рамках сотрудничества разных кросс-функциональных команд;

DSDM — методика разработки динамических систем, основанная на концепции быстрой разработки приложений (RapidApplicationDevelopment, RAD). Представляет собой итеративный и инкрементный подход, который придаёт особое значение продолжительному участию в процессе пользователя/потребителя;

Extremeprogramming (XP) — экстремальное программирование;

Adaptive software development (ADD)	адаптивная разработка программ;
Featuredrivendevelopment (FDD)	разработка, ориентированная на постепенное добавление функциональности;
GettingReal	итеративный подход без функциональных спецификаций, использующийся для веб-приложений;
MSFfogAgileSoftwareDevelopment	гибкая методология разработки ПО компании Microsoft;
Scrum	устанавливает правила управления процессом разработки и позволяет использовать уже существующие практики кодирования, корректируя требования или внося тактические изменения.

Следует отметить, что в чистом виде методологии гибкого программирования редко используются командами разработчиков. Как правило, успешные команды применяют полезные приемы и свойства нескольких процессов, подстраивая их под конкретное представление команды о гибкости процесса разработки.

Вопросы

1. Поясните понятие "гибкая методология разработки программного обеспечения".
2. Какие компетенции необходимы для команды разработчиков, использующих гибкие методологии.
3. Как управляют рисками в гибких методологиях разработки ПО?
4. Какие задачи выполняются на итерациях в методологии гибкой разработки?
5. Назовите ключевые ценности методологий гибкой разработки ПО.

6. Назовите основные принципы гибкой разработки ПО.

7. Какие существуют методологии, которые соответствуют принципам гибкой разработки ПО?

8. Поясните, как в гибком подходе относятся к документированию и выпуску работоспособного кода.

9. Поясните, как должно быть организовано взаимодействие с заказчиком в гибком подходе к разработке ПО.

10. Поясните, как относятся к изменениям в гибком подходе к разработке ПО.

4. Управление жизненным циклом приложений

Введение

Управление жизненным циклом приложений (applicationlifecyclemanagement – *ALM*) – это концепция управления программным проектом на всех этапах его жизни. Для реализации этой концепции компания Microsoft предлагает решение на основе VisualStudio и TeamFoundationServer (TFS). Технологии *ALM* в *Visual Studio* позволяют разработчикам контролировать *жизненный цикл* создания *ПО*, сокращая время разработки, устраняя издержки и внедряя непрерывный цикл реализации бизнес-ценностей.

Управление жизненным циклом приложения в *Visual Studio* базируется на следующих принципах:

- продуктивность (productivity);
- интеграция (integration);
- расширяемость (extensibility).

Продуктивность обеспечивается возможностью совместной работы и управлением сложностью продукта. Все элементы проекта (требования, задачи, тестовые случаи, ошибки, исходный код и построения) и отчеты

централизованно управляются через TFS. Инструменты визуального моделирования архитектуры, возможности управления качеством кода, инструменты тестирования позволяют управлять сложностью продукта.

Интеграция обеспечивается возможностями *Visual Studio* по предоставлению всем участникам проекта информации о состоянии дел, что упрощает коммуникацию между членами команды и обеспечивает прозрачность хода процесса проектирования.

Расширяемость обеспечивается API-интерфейсом служб TFS и интегрированной средой разработки (integrateddevelopmentenvironment – *IDE*). API-*интерфейс* служб TFS позволяет создавать собственные инструменты и расширять существующие, а *IDE* – конечным пользователям и сторонним разработчикам добавлять инструменты с дополнительными функциями.

При создании программного продукта необходимо вначале спроектировать архитектуру, что возлагается на архитектора программного продукта. На основе архитектуры осуществляется разработка, что является предназначением разработчика программного продукта. Созданный продукт необходимо тестировать на его соответствие требованиям заказчика, что осуществляет *тестировщик*. *Visual Studio* и TFS обеспечивают совместную командную работу архитектора, разработчика и тестировщика, предоставляя им необходимый *инструментарий* и функциональные возможности для выполнения требуемых *работ*.

Архитектурное проектирование

В *Visual Studio* для архитектурного проектирования используются инструменты визуального проектирования на основе языка *UML*, которые предназначены для следующего:

- визуализации архитектурных аспектов проектируемой системы;
- создания моделей структуры и поведения системы;
- разработки шаблонов для проектирования системы;

- документирования принятых решений.

Диаграммы *UML* позволяют визуально описывать *приложение*, наглядно представлять архитектуру и документировать требования к приложению.

Архитектурные инструменты в *Visual Studio* 2012 *Ultimate* позволяют создавать шесть видов схем и документ ориентированных графов:

- схема классов UML;
- схема последовательностей UML;
- схема вариантов использования UML;
- схема активности UML;
- схема компонентов UML;
- схема слоев.

Схемы (диаграммы) классов UML описывают объекты в прикладной системе. Диаграммы классов отражают иерархию внутри приложения или системы и связи между ними.

Схемы (диаграммы) последовательностей UML показывают взаимодействие между различными объектами. Они используются для демонстрации взаимодействия между классами, компонентами, подсистемами или субъектами.

Схемы (диаграммы) вариантов использования UML определяют функциональность системы и описывают с точки зрения пользователей их возможные действия с программным продуктом. Данные диаграммы определяют связи между функциональными требованиями, пользователями и основными компонентами системы.

Схемы (диаграммы) активности UML описывают бизнес-процесс или программный процесс в виде потока работ через последовательные действия. Диаграммы активности используются для моделирования логики в конкретном варианте использования или для моделирования подробностей бизнес-логики.

Схемы (диаграммы) компонентов UML описывают распределение программных составляющих приложения, позволяя наглядно отобразить на высоком уровне структуру компонентов и служб. С помощью этих схем можно визуализировать компоненты и другие системы, показывая связи между ними. В качестве компонентов могут выступать исполни-

тельные модули, DLL-библиотеки и другие системы.

Схемы (диаграммы) слоев	используются для описания логической архитектуры системы. Диаграммы слоев могут использоваться для проверки того, что разработанный код отвечает высокоуровневому проекту на схеме слоев. Диаграммы позволяют проверять архитектуру приложения на соответствие базе кода.

Документ ориентированных графов позволяет создать *граф* зависимостей, отображающий отношения между компонентами архитектурных артефактов. Графы зависимостей, обеспечивают визуальные способы проверки кода, анализа зависимостей между файлами.

Разработка приложения

Основным средством разработки в VisualStudio 2012 является интегрированная *среда разработки* (*IDE*). *IDE*-среда интегрирована со средствами модульного тестирования и обеспечивает возможности выявления неэффективного, небезопасного или плохо написанного кода, управление изменениями и *модульное тестирование* как кода, так и *базы данных*.

Важным инструментом разработчика программного обеспечения является *модульное тестирование*, которое реализуется в среде UnitTestFramework. Назначением модульных тестов является проверка того, что код работает правильно с точки зрения программиста. Модульные тесты формируются на более низком уровне, чем другие виды тестирования, и проверяют, работают ли лежащие в их основе функции так, как ожидается. Для модульного тестирования используется метод прозрачного ящика, для которого требуется *знание* внутренних структур кода.

Модульные тесты помогают обнаружить проблемы проектирования и реализации. Кроме того, модульный тест является хорошей документацией по использованию проектируемой системы. Хотя *модульное тестирование* требует дополнительного программирования, но его применение окупается за счет сокращения затрат на отладку приложения.

Модульные тесты являются важным элементом регрессионного тестирования. *Регрессионное тестирование* представляет собой повторное тестирование части программы после внесения в неё изменений или дополнений. Цель регрессионного тестирования – выявление ошибок, которые могут появиться при внесении изменений в программу

В VisualStudio 2012 имеется *функция "Анализ покрытия кода"*, которая проводит *мониторинг* того, какие строки кода исполнялись в ходе модульного тестирования. Результатом анализа покрытия кода является выявление областей кода, которые не покрыты тестами.

Важным аспектом создания качественного программного продукта является соблюдение разработчиками правил и стандартов организации в написания кода. В VisualStudio 2012 имеются *функции анализа кода*, которые позволяют проанализировать код, найти типичные ошибки, нарушения стандартов и предложить меры по устранению ошибок и нарушений. Наборы правил анализа кода поставляются с VisualStudio 2012. Разработчики могут настроить свои проекты на определенный набор правил, а также добавить свои специфичные правила анализа кода.

В процессе анализа кода используются метрики кода, которые дают количественные оценки различных характеристик кода. Метрики позволяют определить сложность кода и его изолированные области, которые могут привести к проблемам при сопровождении приложения. В VisualStudio 2012 используются следующие метрики кода (таблица 4.1):

Таблица 4.1. Метрики кода

сложность организации циклов	- определяет число разных путей кода;
глубина наследования	- определяет число уровней в иерархии наследования объектов;
объединение классов	- определяет число классов, на которые есть ссылки;
строки кода	- определяет количество строк кода в исполняемом методе;
индекс удобства поддержки	- оценивает простоту обслуживания кода.

Для анализа производительности и эффективности использования ресурсов приложением в VisualStudio 2012 имеются инструменты профилирования. *Профилирование* представляет собой процесс наблюдения и записи показателей о поведении приложения. *Инструментарий* профилирования (профилировщики) позволяют обнаружить у приложения проблемы с производительностью. Такие проблемы, как правило, связаны с кодом, который выполняется медленно, неэффективно или чрезмерно использует системную *память*. *Профилирование* обычно используется для выявления участков кода, которые в ходе выполнения приложения выполняются часто или долгое время.

Профилировщики бывают с выборкой и инструментированием. Профилировщики с выборкой делают периодические снимки выполняющегося приложения и записывают его состояние. Профилировщики с инструментированием добавляют маркеры отслеживания в начало и конец каждой исследуемой функции. В процессе работы профилировщика маркеры активизируются, когда *поток* исполнения программы входит в исследуемые функции и выходит из них. *Профилировщик* записывает данные о приложении и о том, какие маркеры были затронуты в ходе исполнения приложения. В VisualStudio 2012 поддерживается *профилирование* с выборкой и с инструментированием. Для анализа производительности необходимо:

- создать сеанс новый сеанс производительности;
- с помощью Обозревателя производительности задать свойства сеансы;
- запустить сеанс, выполняя приложение и профилировщик;
- проанализировать данные в отчетах по производительности.

Большинства корпоративных приложений работает с базами данных, что определяет необходимость разработки и тестировании приложений совместно с базами данных командой проекта. В VisualStudio 2012 имеет-

ся *инструментарий* создания баз данных и развертывания изменений в них. Для этого используется автономная разработка схем баз данных, которая позволяет вносить изменения в схемы без подключения к производственной базе данных. После внесения изменений в среду разработки VisualStudio 2012 позволяет протестировать их в самой среде разработки и/или выделенной среде тестирования. Кроме того, VisualStudio 2012 позволяет сгенерировать псевдореальные данные для проведения тестов. При положительных результатах тестирования VisualStudio 2012 позволяет сгенерировать сценарии для обновления производственной *базы данных*. *Цикл разработки базы данных* приложения состоит из следующих шагов:

- перевод схемы базы данных в автономный режим;
- итеративная разработка приложения с базой данных;
- тестирование схемы базы данных;
- построение и развертывание базы данных и приложения.

Для совершенствования процесса отладки приложений в VisualStudio 2012 имеется *функция* интеллектуального отслеживания работы программы *IntelliTrace*. *Функция IntelliTrace* конфигурируется с помощью следующих разделов:

- Общие (General);
- Дополнительно (Advanced);
- События IntelliTrace (IntelliTrace Events);
- Модули (Modules).

Раздел *Общие* позволяет включить и отключить функцию *IntelliTrace* и задать *запись* только событий либо дополнительной информации, включающей события, данные диагностики, вызовы и отслеживание на уровне методов. В разделе *Дополнительно* задается расположение для генерируемого файла журнала и его максимальный размер. В разделе *События IntelliTrace* перечисляются все события диагностики, которые будут соби-

раться в ходе отладки приложения. Раздел *Модули* определяет *список* модулей, для которых необходимо собирать данные в процессе отладки приложения. При записи событий, происходящих в приложении, происходит их перехват при работе приложения, и *информация* о событиях фиксируется в журнале. При отладке с *IntelliTrace* можно приостановить интерактивный *сеанс* отладки и просмотреть события или вызовы. Также имеется возможность остановки выполнения приложения и пошаговое движение назад и вперед в интерактивном сеансе отладки, а также воспроизведение записанного сеанса отладки.

Тестирование приложения

Тестирование приложения является необходимым этапом управления жизненным циклом приложения. Тестирование выполняет разработчик в процессе создания кода приложения, а также *тестировщик* при проверке качества разрабатываемого программного продукта.

VisualStudio 2012 *Ultimate* предоставляет разработчику инструмент для создания и использования модульных тестов, нагрузочных тестов и веб-тестов производительности, а также тестов пользовательского интерфейса.

Модульные тесты представляют собой низкоуровневые программные тесты, написанные на языках *Visual* C#, *Visual Basic* или *Visual*C++. Они позволяют быстро проверить наличие логических ошибок в методах классов. Основной их целью является проверка того, что код приложения работает так, как ожидает разработчик.

Нагрузочные тесты используются для исследования работоспособности приложения путем моделирования *множества* пользователей, которые работают с программой одновременно. Система позволяет использовать большое количество виртуальных пользователей на локальных и удаленных компьютерах при выполнении нагрузочного теста. В *Visual Studio Ultimate* имеется три встроенных шаблона нагрузки: постоянный,

пошаговый и с учетом эталона. Выбор шаблона нагрузки определяется целям нагрузочного теста.

Тесты пользовательского интерфейса позволяют автоматически сформировать код теста, путем записи действий пользователя при работе с приложением, и впоследствии выполнять эти тесты автоматически.

Архитектура средств тестирования в VisualStudio 2012 приведена на рис. 4.1. Центральное *место* занимает платформа модульного тестирования. *Доступ* к средствам тестирования может осуществляться из обозревателя тестов, командной строки и при построении приложения через TeamBuild. Платформа тестирования обеспечивает подключение плагинов тестирования. В составе VisualStudio 2012 установлены *плагины* MS-TestManaged и MS-TestNative. *Плагины* сторонних разработчиков (NUnit, xUnit.*net*, MbUnit и другие) можно подключить к платформе юнит-тестирования.

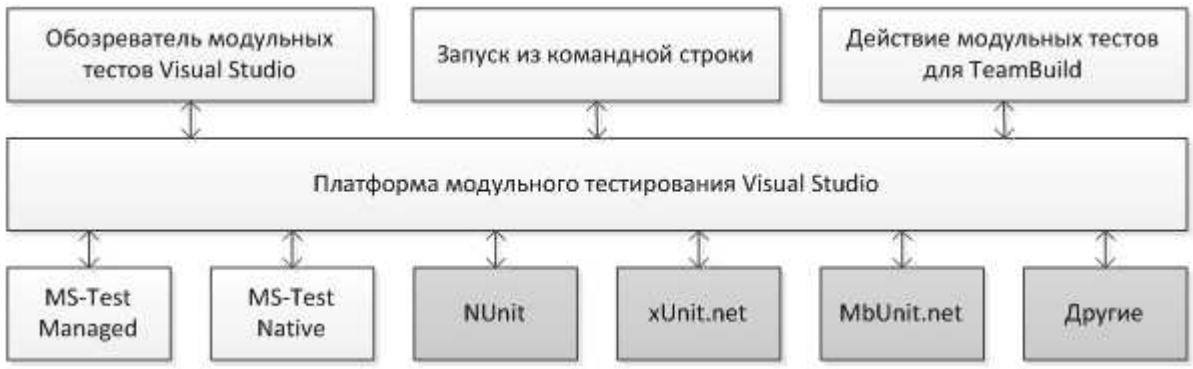

Рис. 4.1. Архитектура инструментов тестирования

Средства модульного тестирования ориентированы на использование разработчиками в процессе написания кода.

Для тестировщиков в VisualStudio 2012 *Ultimate* имеется специализированный инструмент – MicrosoftTestManager, который позволяет создавать планы тестирования, формировать, добавлять и удалять тестовые случаи, определять и управлять физическими и виртуальными тестовыми средами, выполнять ручные и автоматические тесты.

Создание эффективной среды тестирования сложных приложений выполняется с помощью лаборатории тестирования – LabManagement, которая интегрирована с TeamFoundationServer 2012, и представляет собой *диспетчер* виртуальной среды. Лаборатория тестирования предоставляет следующие возможности:

- создание, управление и освобождение сред, состоящих из одной или более виртуальных машин;
- автоматическое развертывание построений в виртуальных средах;
- выполнение ручных и автоматических тестов в виртуальных средах;
- использование снимков, позволяющих быстро вернуть среды к заданному состоянию;
- сетевая изоляция виртуальных сред.

Администрирование виртуальной среды LabManagement производит *диспетчер* MicrosoftSystemCenterVirtualMachineManager (SCVMM), с помощью которого производят необходимые настройки виртуальной тестовой лаборатории.

В VisualStudio 2012 и TeamFoundationServer добавлено средство взаимодействия с пользователями разработанных программных продуктов, которое по запросу позволяет сформировать отзыв пользователя в базе данных для последующей обработки командой проекта.

Вопросы

1. Что определяет понятие "управление жизненным циклом приложений"?
2. Назовите принципы управления жизненным циклом приложения в Visual Studio.

3. Для чего предназначены инструменты визуального проектирования в Visual Studio?

4. Какие виды схем архитектурного проектирования можно подготовить в Visual Studio 2012?

5. Для чего предназначена функция "Анализ покрытия кода" в VisualStudio 2012?

6. Поясните назначение профилировщика в Visual Studio 2012.

7. Поясните назначение программы IntelliTrace.

8. Какие метрики кода используются в VisualStudio 2012?

9. Какое назначение нагрузочных тестов в Visual Studio 2012?

10. Какие возможности предоставляет лаборатория тестирования - LabManagement?

5. Архитектура и функциональные возможности Visual Studio Team Foundation Server

Введение

MicrosoftVisualStudio *TeamFoundationServer(TFS)* предназначен для обеспечения совместной работы команд разработчиков программного обеспечения. TeamFoundationServer предоставляет следующие функциональные возможности:

- управление проектами;
- отслеживание рабочих элементов;
- контроль версий;
- управление тестовыми случаями;
- автоматизация построения;
- отчетность.

Архитектура TeamFoundationServer 2012 является трехуровневой сервис-ориентированной (рис. 5.1). Уровень приложения поддерживается

веб-сервером *ASP. NET*, размещенном в среде *IIS*. Уровень данных поддерживается сервером баз данных MSSQLServer 2012.

Рис. 5.1. Архитектура TeamFoundationServer

Team Foundation Server представляет с логической точки зрения веб-*приложение*, состоящее из нескольких веб-служб, выполняющихся на *уровне приложения*. Данные службы реализуют функциональность TFS. В состав веб-служб уровня приложения входят: управление версиями, служба построения, отслеживания рабочих элементов, службы платформы TFS и лаборатории тестирования LabManagement. Серверная *объектная модель* является интерфейсом прикладного программирования для TFS. При необходимости расширение функциональности TFS целесообразно строить на базе серверной объектной модели.

Уровень данных состоит из нескольких реляционных баз данных и хранилища данных: *базы данных* конфигурации сервера, *базы данных* аналитики, *базы данных* коллекции командных проектов и хранилища данных. *Информация* командных проектов сохраняется в реляционных базах данных и через запланированные интервалы времени помещается в *хранилище данных*.

Клиентский уровень может реализовываться в оболочке VisualStudio и веб-браузере. Клиентский уровень взаимодействует с уровнем приложе-

ний через серверную объектную модель и использует веб-службы. Кроме того, клиентский уровень включает интеграцию с Microsoft Office.

Развертывание Team Foundation Server

Для *Team Foundation Server* можно выполнить *развертывание* несколькими способами: на одном сервере; на нескольких серверах; в одном домене, рабочей группе или в нескольких доменах.

В простейшей серверной топологии для размещения компонентов, составляющих логические уровни *Team Foundation*, используется один физический *сервер* (рис. 5.2). При установке TFS с одним сервером все компоненты (*приложение*TeamFoundationServer, SQLServer, ReportingServices и WidowsSharePointServices) устанавливаются на одном компьютере. Такая *конфигурация* предполагает выполнение построения (TeamFoundation Build) и тестирование либо на сервере, либо на клиентских компьютерах. Общее число пользователей для такой конфигурации, как правило, не более 50.

Рис. 5.2. Простейшая серверная топология TFS

В простой серверной топологии для размещения компонентов, составляющих логические уровни *Team Foundation*, также используется один физический *сервер* (рис. 5.3). Однако в такой технологии учитывается так-

же дополнительная нагрузка на процессорные мощности, создаваемая программным обеспечением для построения и тестирования.

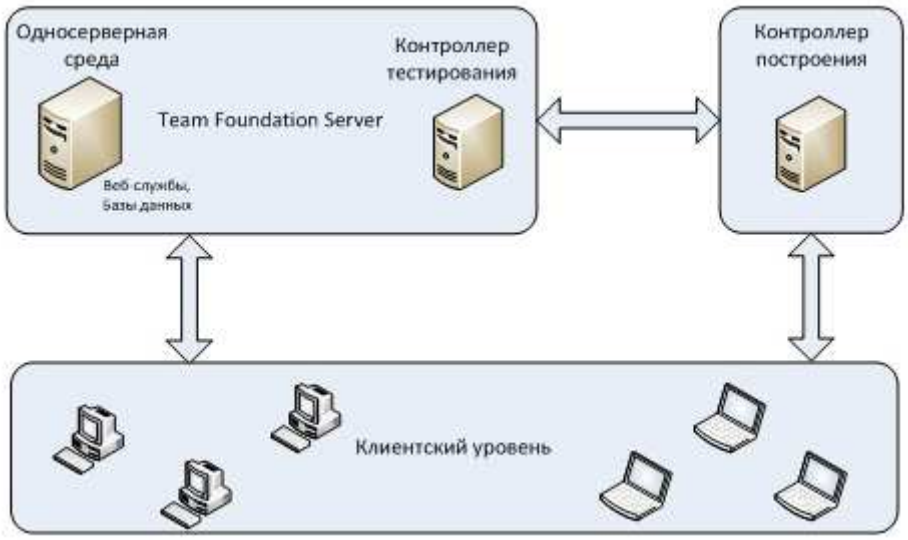

Рис. 5.3. Простая серверная топология TFS

На рис. 5.3 веб-службы и *базы данных* для *Team Foundation* размещаются на одном физическом сервере, но службы построения устанавливаются на отдельный *компьютер*. К *Team Foundation Server* можно получить *доступ* с клиентских компьютеров, принадлежащих к той же рабочей группе или тому же домену. В данной конфигурации *контроллер* построения для выполнения *TeamFoundationBuild* и *контроллер* тестирования устанавливаются на отдельных компьютерах. Общее число пользователей для такой конфигурации, как правило, не более 100.

В серверной топологии средней сложности используются два или больше серверов для размещения логических компонентов на уровне данных и приложений *Team Foundation* (рис. 5.4). На рис. 5.4 службы уровня приложений для *Team Foundation Server* развертываются на одном сервере, а *базы данных* для TFS устанавливаются на отдельном сервере. На отдельных серверах размещаются веб-приложение SharePoint и экземпляр служб отчетов *SQL Server*. Портал для каждого командного проекта размещается в веб-приложении SharePoint. Сервер *Team Foundation Build* и тестовые

контроллеры команды развертываются на дополнительных серверах. Общее число пользователей для такой конфигурации, как правило, не более 1000.

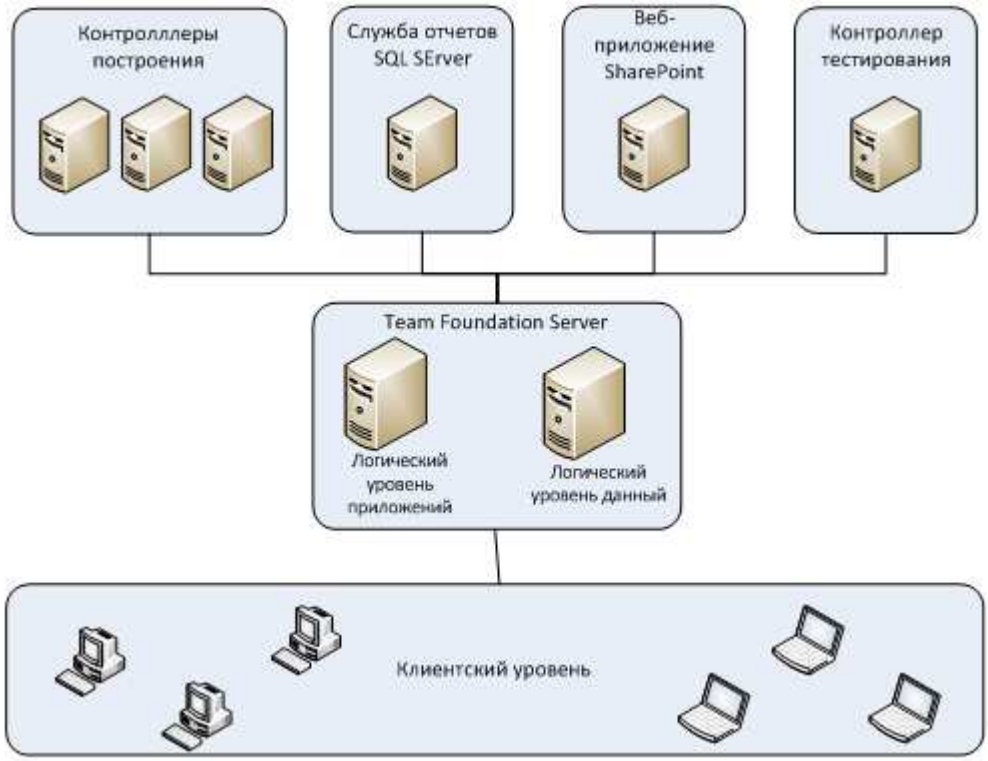

Рис. 5.4. Серверная топология TFS средней сложности

Для управления командной разработкой TeamFoundationServer содержит одну или несколько коллекций командных проектов, каждая из которых может содержать ноль или более проектов.

Шаблоны командных проектов

Командный проект представляет коллекцию рабочих элементов, кода, тестов и построений, которые охватывают все артефакты, используемые в жизненном цикле программного проекта [6]. Командный проект строится на основе шаблона, который представляет набор XML-файлов, содержащих детали того, как должен осуществляться процесс. В TFS 2012 имеются следующие шаблоны проектов:

- MSFforCMMIProcessImprovement 6.0, который предназначен для больших команд со строго формальным подходом к управлению проектами на основе модели CMM/CMMI;
- MSFforAgileSoftwareDevelopment 6.0, который определяет гибкий подход к управлению проектами разработки программного обеспечения;
- MicrosoftVisualStudioScrum 2.2., который предназначен для небольших команд (до 7 - 10 участников), которые используют гибкую методологию и терминологию Scrum.

При создании командного проекта предоставляется возможность настраивания и управления следующими областями проекта:

- отслеживание рабочих элементов позволяет определить начальные типы рабочих элементов, общие и пользовательские запросы, создать начальные рабочие элементы;
- классификации позволяют определить начальные области и итерации проекта;
- веб-сайт на базе SharePoint позволяет управлять библиотекой документов проекта;
- система контроля версий позволяет определять начальные группы безопасности, разрешения, правила возврата версий;
- отчеты формируются в папки, создаваемые на сайте командного проекта;
- группы и разрешения включают группы безопасности TFS и разрешения для каждой группы;
- построения включают стандартные процессы построения для создания новых построений;
- лаборатория включает стандартные процессы лаборатории для использования, а также разрешения лаборатории для каждой группы;
- управление тестированием включает стандартные конфигурации тестирования, переменные, параметры и состояния разрешений.

Рабочими элементами в TeamFoundationServer являются: пользовательское описание функциональности, задачи, тестовые случаи, ошибки и препятствия. Этими элементами нужно управлять для выполнения программного проекта. Система отслеживания рабочих элементов позволяет создавать рабочие элементы, отслеживать их состояние, формировать историю их изменения. Все данные по рабочим элементам сохраняются в базе данных TFS.

TeamFoundationServer включает централизованную систему контроля версий. *Система контроля версий* TFS предоставляет следующие возможности:

- атомарные возвраты. Изменения, вносимые в файлы, упаковываются с "набор изменений". Возврат файла в наборе изменений представляет собой неделимую транзакцию. Этим гарантируется согласованность базы кода;
- ассоциация операций возврата с рабочими элементам, что позволяет отслеживать требования от начальной функции до задач и возврата в систему контроля версий кода, реализующего эту функцию;
- ветвление и объединение при разработке кода;
- наборы отложенных изменений позволяют создавать копии кода, не записывая их в главное хранилище системы контроля версий;
- использование подписей позволяет пометить набор файлов в определенной версии с помощью текстовой подписи;
- одновременное извлечение позволяет нескольким членам команды одновременно редактировать файл с последующим объединением изменений;
- отслеживание истории файлов;
- политики возврата, которые предоставляют возможность выполнить код для проверки допустимости возврата;

• примечания при возврате позволяют формировать метаданные о возврате;

• прокси-сервер позволяет оптимизировать работу с региональными центрами разработки.

Построение проекта программного продукта в TFS реализует *Сервер Team Foundation Build.* Он позволяет стандартизировать инфраструктуру построений для команды проекта. Построение может выполняться в следующих режимах:

• ручной, при котором построение вручную ставится в очередь построений;

• непрерывная интеграция, которая позволяет выполнять построение при каждом возврате в систему контроля версий;

• прокрутка построений, при которой возвраты группируются вместе, чтобы в построение включались изменения за определенный период времени;

• условный возврат, при котором возвраты принимаются лишь в том случае, если успешно прошли слияния и построения отправленных изменений;

• расписание, которое позволяет настроить время построения на определенные дни недели.

Разработчик имеет возможность взаимодействовать с ключевыми службами TeamFoundationServer посредством:

• командного обозревателя (TeamExplorer) VisualStudio 2012, который предоставляет доступ к функциям управления жизненным циклом приложений, включая командные проекты, Мои работы, анализ кода, управление версиями и построениями;

• доступа через веб (Team Web Access), посредством которого предоставляется веб доступ к функциям управления жизненным циклом прило-

жений, включая командные проекты, рабочие группы, управление проектами, управление версиями и построениями;

- MicrosoftExcel, посредством которого предоставляется возможность определять и изменять рабочие элементы массивом, а также создавать отчеты на основе запроса рабочего элемента;
- MicrosoftProject, посредством которого предоставляется возможность управлять планом проекта, задачами расписания, ресурсами, формировать календарь проекта, диаграммы Ганта и представления ресурсов;
- консоли администрирования TeamFoundationServer, посредством которой осуществляется настройка TFS;
- инструментов командной строки;
- сторонних средств интеграции.

Вопросы

1. Поясните назначение и функциональные возможности MicrosoftVisualStudio 2012 TeamFoundationServer.
2. Дайте характеристику архитектуры TFS.
3. Дайте характеристику простейшей серверной топологии TFS.
4. Дайте характеристику простой серверной топологии TFS.
5. Дайте характеристику серверной топологии TFS средней сложности.
6. Что представляет собой командный проект в TFS.
7. Какие шаблоны проектов имеются в TFS 2012?
8. Какие рабочие элементы определены в TeamFoundationServer.
9. Какие возможности предоставляет система контроля версий TFS?
10. Какие возможности имеет разработчик для взаимодействия с ключевыми службами TeamFoundationServer?

6. Организация командной разработки на базе Visual Studio и Team Foundation Server

Введение

При командной разработке программного обеспечения важными вопросами являются планирование работ, составление расписания, управление областью проекта, коммуникации, составление отчетов, анализ и постоянное совершенствование процесса [6]. Для решения этих вопросов TeamFoundationServer предлагает следующие инструменты:

- *шаблон процесса*, который определяет процесс, используемый командным проектом;
- *руководство по процессу*, которое содержит описание шаблонов процесса;
- *коллекция командных проектов*, которая представляет собой контейнер для нескольких командных проектов;
- *командный проект*, который хранит и организует данные обо всем жизненном цикле разработки программного обеспечения;
- *отслеживание рабочих элементов*, которое позволяет отслеживать состояние рабочих элементов и другую информацию, связанную с ними;
- *портал проекта/панели мониторинга*, на которых предоставляется информация по проекту для всех членов команды;
- *элементы* планирования для управления списками требований как на уровне проекта, так и на уровне итерации;
- *отчетность*, которая позволяет формировать отчеты в ходе жизненного цикла проекта;
- *интеграция* с MicrosoftProject и MicrosoftExcel для управления проектами из среды MicrosoftOffice.

Командный проект

Разработка программного обеспечения начинается с создания командного проекта. Командный проект содержит информацию о каждом шаге жизненного *цикла* разработки программного обеспечения, включая требования пользователей, задачи, тестовые случаи, ошибки, препятствия, построения.

При создании командного проекта необходимо задать его имя, описание, определиться с шаблоном процесса, требованиями к системе контроля версий и необходимостью создания портала для проекта.

В результате создания командного проекта TeamFoundationServer формирует следующие папки:

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 создания командного проекта руководитель формирует команду. Для каждого члена команды *руководитель проекта* определя-

ет *доступ* к проекту в целом и отдельным артефактам, которые важны для работы каждого члена команды.

Для структурирования проекта используются области и итерации. *Области* могут определяться исходя из определенного функционального назначения этапа *работ*, а *итерации* – в виде набора *работ* на заданном временном интервале.

Рабочие элементы

При планировании командного проекта ключевыми сущностями являются рабочие элементы. В зависимости от шаблона командного проекта набор и наименование рабочих элементов несколько отличаются. Так для гибкой методологии Agile рабочими элементами являются [6]:

- Пользовательское описание функциональности (UserStory);
- Задача (Task);
- Ошибка (Bug);
- Препятствие (Issue);
- Тестовый случай.

Рабочий элемент "Пользовательское описание функциональности" представляет собой пользовательское требование, которое необходимо выполнить при реализации проекта.

Рабочий элемент "Задача" создается в проекте для назначения и выполнения работы. Задачи предоставляют детали реализации для пользовательских требований.

Рабочий элемент "Ошибка" используется для отслеживания и мониторинга проблем в программном продукте.

Рабочий элемент "Препятствие" используется для фиксации в проекте событий или объектов, которые создают проблемы в выполнении проекта и должны быть устранены в ходе текущей или будущей итерации.

Рабочий элемент "Тестовый случай" описывает условия проверки правильности выполнения программным продуктом требований пользователя.

Рабочие элементы могут включать наименование, описание назначения, состояние, кому назначено, ценность для бизнеса, принадлежность к рабочей области.

При планировании командного проекта *пользовательские требования* записываются в *Элемент "Невыполненная работа по продукту*". Данную работу можно проводить с использованием:

- Командного обозревателя в VisualStudio;
- Microsoft Project;
- Microsoft Excel.

Пользовательские описания функциональности могут быть связаны между собой, а также с дочерними или родительскими элементами. В качестве дочерних элементов могут быть, например, задачи или ошибки.

Разработка программного кода

Задачи назначаются разработчикам программного кода. Разработчики проводят *кодирование* и *модульное тестирование* задач в среде VisualStudio. Разрабатываемые коды находятся под контролем системы управления версиями в хранилище кода. Работоспособные коды задач собираются в построении с помощью TeamServiceBuild и передаются в систему управления версиями на TeamFoundationServer. В процессе работы разработчик проводит извлечение (checkingout) файлов с кодами задач приложения в свою рабочую область на инструментальном компьютере. После выполнения определенного этапа работ разработчик осуществляет возврат (checkingin) в хранилище TFS кода. При возврате кода формируется набор изменений (changeset), который содержит всю информацию, связанную с возвратом (ссылки на рабочие элементы, исправления, примечания, политики и данные о владельце, дате и времени). Это дает возмож-

ность разработчикам просматривать различные версии файлов и анализировать вносимые изменения.

Тестирование

Тестировщики на основе тестовых случаев подготавливают тесты для заданных пользовательских требований. Для тестирования файлы кода извлекаются из хранилища TFS и подвергаются тестированию. В случае выявления ошибки формируется рабочий элемент "Ошибка", который направляется разработчику для исправления. Если тесты проходят, то соответствующее пользовательское требование считается выполненным и для него устанавливают состояние "Готово".

Отчеты

TeamFoundationServer 2012 имеет мощную систему сбора информации по ходу проекта и подготовки отчетов. Отчеты предназначены как для руководителей проекта, так и для членов команды. Система отчетности базируется на *хранилище данных* TFS. Данные могут быть представлены в виде отчетов, которые позволяют просматривать метрики проекта. Система отчетов позволяет отслеживать рабочие элементы, построения, статистику системы контроля версий, результаты тестов, индикаторы качества проекта. TeamFoundationServer предоставляет набор отчетов, но имеется возможность создавать и пользовательские отчеты.

В TFS имеются три хранилища данных:

- операционное хранилище;
- хранилище данных;
- OLAP-куб.

Операционное хранилище TFS представляет собой набор реляционных баз данных для хранения информации, такой как исходный код, отчеты построения, результаты тестов и отслеживание рабочих элементов.

Хранилище данных TFS предназначено для выполнения запросов и создания отчетов. *Хранилище данных* получает данные из операционного

хранилища через определенные промежутки времени. *Хранилище данных* концептуально построено по схеме "*Звезда*".

OLAP-куб TeamFoundationServer является многомерной базой данных, которая содержит агрегированные данные для подготовки аналитических отчетов. OLAP-куб получает данные из хранилища данных TFS через запланированные интервалы времени. Данные многомерной *базы данных* могут использоваться различными клиентскими приложениями, включая MicrosoftExcel и *конструктор* SQL-отчетов. TeamFoundationServer включает два набора отчетов: отчеты MicrosoftExcelReports и отчеты службы отчетности SQLReportingServicesReports. MicrosoftExcel позволяет создавать отчеты из OLAP-куба TFS, либо используя запросы рабочих элементов проекта. Отчеты могут быть опубликованы на SharePoint-портале проекта.

Вопросы

1. Какие инструменты имеются в TeamFoundationServer для управления командными проектами?
2. Какие папки генерируются при создании командного проекта?
3. Для чего используются в проекте области и итерации?
4. Какие рабочие элементы имеются в шаблоне Agile?
5. С помощью каких клиентских инструментов можно проводить планирование командного проекта?
6. Поясните процесс создания кода разработчиком командного проекта в среде VisualStudio.
7. Поясните процесс тестирования программных продуктов командного проекта.
8. Какие хранилища данных имеются в TFS?
9. Для чего предназначено операционное хранилище TFS?
10. Для чего предназначено хранилище данных TFS?

7. Обеспечение качества программных продуктов

Введение

Качество программного продукта определяется по нескольким критериям. Качественный *программный продукт* должен отвечать функциональным и нефункциональным требованиям, в соответствии с которыми он создавался, иметь ценность для бизнеса, отвечать ожиданиям пользователей.

В жизненном цикле управления приложениями качество должно отслеживаться на всех этапах жизненного *цикла ПО*. Оно начинает формироваться с определения необходимых требований. При задании требований необходимо указывать желаемую функциональность и способы проверки её достижения.

Качественный *программный продукт* должен обладать высоким потребительским качеством, независимо от области применения: внутреннее использование разработчиком, бизнес, наука и образование, медицина, коммерческие продажи, социальная сфера, развлечения, веб и др. Для пользователя *программный продукт* должен удовлетворять определенному уровню его потребностей.

Важным аспектом создания качественного *ПО* является обеспечение нефункциональных требований, таких как удобство в эксплуатации, *надежность*, *производительность*, защищенность, *удобство сопровождения*. *Надежность ПО* определяет способность без сбоев выполнять заданные функции в заданных условиях и в течение заданного отрезка времени. *Производительность* характеризуется временем выполнения заданных транзакций или длительных операций. Защищенность определяет степень безопасности системы от повреждений, утраты, несанкционированного доступа и преступной деятельности. *Удобство сопровождения* определяет легкость, с которой обслуживается продукт в плане про-

стоты исправления дефектов, внесения корректив для соответствия новым требованиям, управления измененной средой.

Управление жизненным циклом программного продукта помогает разработчикам целенаправленно добиваться создания качественного *ПО*, избегать потерь времени на переделку, повторное проектирования и перепрограммирование *ПО*.

Тестирование программного обеспечения

Тестирование программного продукта позволяет на протяжении всего жизненного *цикла ПО* гарантировать, что программные проекты отвечают заданным параметрам качества. Главная цель тестирования – определить отклонения в реализации функциональных требований, обнаружить ошибки в выполнении программ и исправить их как можно раньше в процессе выполнения проекта.

На протяжении всего жизненного цикла разработки *ПО* применяются различные типы тестирования для гарантии того, что промежуточные версии отвечают заданным показателям качества. При этом применяются автоматические и ручные тесты.

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.

Для разработчика программного обеспечения в VisualStudio 2012 предоставлена возможность создавать модульные и нагрузочные тесты, а также тесты пользовательского интерфейса. В VisualStudio 2012 имеются следующие шаблоны тестовых проектов:

- проект модульного теста, который позволяет создавать модульные тесты в процессе разработки;
- проект с веб-тестами производительности и нагрузочными тестами;
- проект с закодированными тестами пользовательского интерфейса.

Инструментарием тестировщика в VisualStudio 2012 является MicrosoftTestManager (MTM). MTM предназначен для управления жизненным циклом тестирования программного обеспечения, включая планирование, тестирование и *мониторинг*. MTM интегрирован с TeamFoundationServer. С помощью Microsoft TestManager тестировщики подготавливают планы тестирования, управляют тестированием. При создании плана тестирования в него добавляются наборы тестов, тестовые случаи и конфигурации, необходимые для тестирования. Конфигурации используются для установления среды, в которой будут исполняться наборы тестов. Microsoft TestManager позволяет выполнять ручные и автоматические тесты, а также исследовательские тесты. Результаты тестирования сохраняются в базе данных, что позволяет подготавливать различные аналитические отчеты. Ошибки, выявленные в процессе тестирования, фикси-

руются, документируются и передаются разработчикам для их устранения. При внесении изменений в код программной системы возникает необходимость в регрессионном тестировании, причем MTM автоматически формирует план регрессионного тестирования, выявляя какие тесты должны быть повторно выполнены.

Для тестировщиков и разработчиков программного обеспечения VisualStudio 2012 включает *диспетчер* виртуальной среды Lab Management. *Инструментарий* тестирования LabManagement позволяет создать инфраструктуру, которая максимально близко эмулирует реальную среду планируемого использования программного продукта. Такие среды могут использоваться для выполнения автоматических построений, автоматизации тестов и выполнения разработанного кода.

Рефакторинг

Качество кода программного продукта во многом определяется тем, насколько трудно или легко вносить изменения в код, а также тем насколько доступен код для понимания. Созданное программное *приложение* может выполнять требуемые функции, но иметь проблемы с внесением изменений или пониманием созданного кода. В этом случае такое *ПО* нельзя назвать качественным, так как на этапе его сопровождения могут возникнуть проблемы с его модификацией при изменении пользовательских требований.

Для улучшения качества кода программных приложений применяют рефакторинг. По определению Фаулера М., рефакторинг определяется как "процесс изменения программной системы таким образом, что её внешнее поведение не изменяется, а внутренняя структура улучшается". Следовательно, в процессе проектирования для создания высококачественного программного продукта необходимо не только обеспечить выполнение функциональных требований, но и нефункциональных, в частности, удобства сопровождения, что предполагает возможность и простоту внесения

изменений в код, а также возможность легкого понимания созданного кода.

Некачественный *дизайн* кода определяется по ряду признаков:

- жесткость – характеристика программы, затрудняющая внесение изменений в код;
- хрупкость – свойство программы повреждаться во многих местах при внесении единственного изменения;
- 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;
- ненужная сложность характеризуется тем, что программа содержит элементы, которые не используются в текущий момент;
- ненужные повторения, которые состоят в повторяющихся фрагментах кода в программе;
- непрозрачность, которая характеризует трудность кода для понимания.

Для создания качественного дизайна кода целесообразно применять некоторые принципы и паттерны проектирования *ПО* [5, 6].

Принцип единственной обязанности определяет, что у класса должна быть только одна причина для изменения. Из этого принципа следует, что классу целесообразно поручать только одну обязанность.

Принцип открытости/закрытости определяет, что программные сущности (классы, модули, функции) должны быть открыты для расширения, но закрыты для модификации.

Принцип подстановки Барбары Лисков (Liskov Substitution Principle) определяет, что должна быть возможность вместо базового класса подставить любой его *подтип*.

Принцип инверсии зависимостей определяет два положения:

• модули верхнего уровня не должны зависеть от модулей нижнего уровня. И те, и другие должны зависеть от абстракций;

• абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций.

Принцип разделения интерфейсов определяет, что клиенты должны знать только об абстрактных интерфейсах, обладающих свойством сцепленности.

Паттерны проектирования предлагают универсальные, проверенные практикой решения. В [7] приведен обширный перечень паттернов. Среди общего списка паттернов следует выделить те, которые целесообразно применять при гибкой разработке программного обеспечения. Это паттерны *Команда*, Стратегия, Фасад, Посредник, Одиночка, Фабрика, Компоновщик, Наблюдатель, Абстрактный сервер/адаптер/*шлюз*, Заместитель и *Шлюз*, Посетитель и Состояние.

Использование паттернов при разработке позволяет создавать *программное обеспечение*, которое легко модифицировать и сопровождать.

Следует отметить, что для проведения рефакторинга необходимо иметь надежные тесты, которые обеспечивают *контроль* соблюдения функциональных требований при улучшении дизайна кода программного обеспечения.

Вопросы

1. Какими характеристиками должен обладать качественный программный продукт?
2. Какие нефункциональные требования определяют качество программного продукта?
3. Какая роль тестирования в обеспечении качества программного продукта?

4. Какие типы тестов используют для проверки качества программного продукта?

5. Для чего применяется регрессионное тестирование?

6. Какие шаблоны тестовых проектов имеются в VisualStudio 2012?

7. Для чего применяется MicrosoftTestManager? Какие он имеет функциональные возможности?

8. Для чего используется LabManagement при тестировании?

9. Для чего применяют рефакторинг кода?

10. Приведите признаки некачественного кода.

8. Методология гибкой разработки SCRUM

Введение

Методология Scrum представляет собой итеративный процесс разработки программного обеспечения. При такой разработке для программного продукта создается много последовательных выпусков, в которых постепенно добавляется требуемая функциональность. Итеративный подход позволяет по завершению текущей итерации продемонстрировать заказчику работоспособный *программный продукт*, возможно с ограниченной функциональностью, получить отзыв, замечания и дополнительные требования, которые будут учтены в следующих итерациях. Основными артефактами в методологии Scrum являются рабочие элементы, отчеты, книги и панели мониторинга.

Рабочие элементы

Рабочие элементы используются для отслеживания, наблюдения за состоянием хода разработки *ПО* и создания отчетов. Рабочий элемент – это *запись*, которая создается в *Visual* Studio *Team Foundation Server* для задания определения, назначения, приоритета и состояния элемента рабо-

ты. Для шаблона MicrosoftVisualStudioScrum 2.2. определяет следующие типы рабочих элементов:

- невыполненная работа по продукту;
- ошибка;
- задача;
- препятствие;
- тестовый случай.

В методологии Scrum *пользовательские требования*, которые определяют функциональность продукта, задаются *элементами задела работы продукта* (ProductBacklogItem – PBI). Элементы задела работы продукта, которые будем называть "*элементы работы – ЭР*", представляют собой краткое описание функций продукта и оформляются в произвольной форме в виде кратких заметок. Вначале задаются наиболее важные и понятные всем *пользовательские требования* - ЭР. Элементы работы могут детализироваться в виде задач. В процессе создания программного продукта ЭР могут уточняться, добавляться или удаляться из списка требований.

Цикл выпуска продукта в Scrum состоит из ряда итераций, которые называются *спринтами*. Спринт имеет фиксированную длительность, как правило, 1-4 недели. Элементы работы, включенные в очередной спринт, не подлежат изменению до его окончания. Хотя спринт завершается подготовкой работоспособного программного продукта, его текущей функциональности может быть недостаточно для оформления выпуска, имеющего ценность для заказчика. Поэтому выпуск работоспособного программного продукта, который заказчик может использовать, включает, как правило, несколько спринтов.

Организация команды

Организация команды в методологии Scrum определяет три роли:

- владелец продукта (Product owner);
- руководитель (ScrumMaster);

- члены команды (Team members).

Владелец продукта отвечает за всё, что связано с потребительскими качествами программного продукта. Он определяет *пользовательские требования* – ЭР, анализирует их реализацию, обладает правом изменения требований, контролирует качество продукта. Он может быть представителем заказчика в команде или членом команды разработчиков, который представляет интересы заказчика. Владелец продукта выполняет следующие основные задачи:

- определение и приоритезация требований/функций, то есть элементов работ и задач;
- планирование спринтов и выпусков;
- тестирование требований/функций.

Руководитель отвечает за состояние и координацию проекта, продуктивность команды и устранение препятствий, мешающих проекту. В обязанности руководителя входит:

- проведение ежедневных Scrum-собраний;
- привлечение сотрудников вне команды:
- стимулирование эффективного общения членов команды;
- определение размера команды.

Члены команды отвечают за разработку программного продукта высокого качества. Они должны обладать навыками в проектировании и архитектуре программного продукта, бизнес-анализе, программировании, тестировании, настройке баз данных и проектировании пользовательского интерфейса. Члены команды участвуют в планировании спринтов. *Команда* может включать опытных разработчиков и новичков, которые в процессе работы должны совершенствоваться при обмене знаниями с другими членами команды. Члены команды отвечают за следующие задачи в проекте:

- обязательное выполнение элементов работ, включенных в текущий спринт;
- акцент на взаимосвязанных задачах спринта;
- совершенствование команды.

Жизненный цикл проекта ПО

Инструментальная и методическая *поддержка* гибкого подхода к созданию программных продуктов Scrum, реализованная в VisualStudio 2012, позволяет управлять жизненным циклом проекта *ПО* (рис. 8.1) [7].

Рис. 8.1. Жизненный цикл проекта ПО

В начале проектирования владелец продукта и заказчик формируют концепцию программного продукта, которая показывает, для кого предназначен продукт, какие преимущества получат пользователи, и какие существуют конкуренты. Концепция продукта связывается с областью проекта и ограничениями. Область проекта определяет масштаб *работ*, а ограничения – условия, которыми будут руководствоваться для первых спринтов и выпусков. Далее владелец продукта создает *список* всех потенциальных функций продукта – "Невыполненная работа по продукту" (ProductBacklog). *Невыполненная работа по продукту*, которую в даль-

нейшем будем называть *невыполненная работа – НВР*, содержит *список элементов работы* – пользовательских описаний функциональности.

Управление *невыполненной работой* по проекту сводится к поддержанию *элементов работ* в актуальном состоянии. Отдельные элементы *работ* списка НВР могут добавляться или удаляться в процессе создания *ПО*. Это является результатом того, что *команда* получает дополнительную информацию о новых требованиях заказчика к проектируемому программному продукту, а заказчик выясняет, как реализуются его ожидания.

Для элементов *работ* владелец продукта совместно с командой проекта расставляет приоритеты. При планировании спринта в него включают наиболее значимые, с точки зрения владельца продукта, *пользовательские требования* – ЭР, которые характеризуются наибольшей потребительской ценностью. Выбранные элементы *работ* перемещают в *список* "*Незаконченная работа*" (SprintBacklog). *Список* "*Незаконченная работа*" *(НЗР)* отражает состав *работ* планируемого спринта. *Список* НЗР является результатом процесса планирования спринта.

Координацией *работ* в спринте занимается руководитель спринта (ScrumMaster). Он организовывает процесс приоритезации задач спринта, распределения задач между членами команды. Руководитель спринта проводит собрание по планированию *работ*, ежедневные собрания для краткого обсуждения результатов работы и проблем, обзорные собрания в конце спринта и выпуска.

Ежедневные Scrum-собрания имеют продолжительность 15 - 30 минут. Целью таких собраний является выявление проблем, которые тормозят процесс разработки, и определить действия по их нейтрализации. Для простых проблем принимается решение по их устранению, а сложные проблемы откладываются на последующие спринты. В ходе ежедневного Scrum-собрания руководитель задает темп спринта, акцентирует внимание

команды на наиболее важных элементах списка невыполненных *работ*. Каждый член команды сообщает, что было сделано вчера, что будет делать сегодня, и какие имеются препятствия в работе. Если на ежедневном Scrum-собрании возникают вопросы, для решения которых необходимы специалисты, которых в команде нет, тогда руководитель берет на себя *анализ* и возможные пути разрешения данного вопроса.

Результатом спринта является работоспособное *ПО*, возможно обладающее только частью необходимых функций программного продукта. Выпуск спринта может быть развернут у заинтересованных лиц для предварительного анализа соответствия ожиданиям заказчика. Результатом отзыва является формирование "Отзыв" (FeedBack), что может привести к изменения содержания списка "Элемент задела работы продукта". После выполнения всех работ по программному продукту, то есть обнуления списка требований в списке "Элемент задела работы продукта" подготавливается финальный выпуск программного продукта.

Управление невыполненной работой

Список "*Невыполненная работа по продукту*" является одним из ключевых артефактов в методологии Scrum. Успех Scrum-команды во многом определяется качественным содержанием данного списка. *Список* НВР обычно включает пользовательские описания функциональности – элементы работы, а также может включать нефункциональные требования. Для создания списка НВР в TFS могут применяться различные клиентские сервисы (рис. 8.2):

- командный обозреватель Visual Studio;
- веб-доступ черезTeam Web Access;
- Microsoft Office Excel;
- Microsoft Office Project.

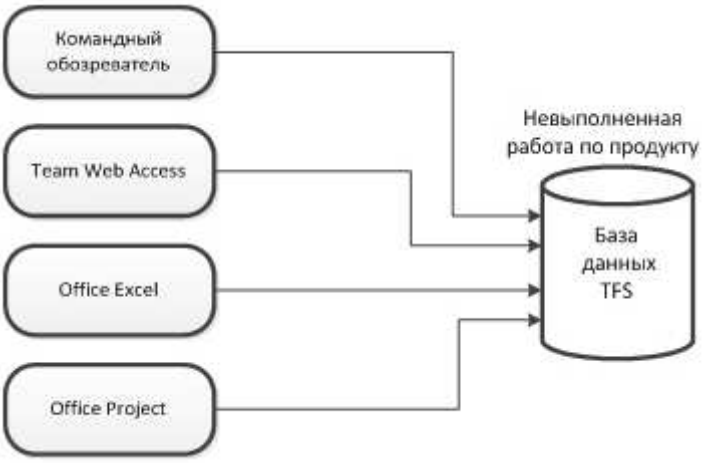

Рис. 8.2. Клиентские сервисы TFS

Владелец продукта на основе требований и пожеланий клиентов формирует *список* функций продукта в виде элементов задела работы продукта, которые помещает в *список* "*Невыполненная работа по продукту*". При создании нового элемента невыполненной работы для него устанавливается состояние "Новый" (рис. 8.3).

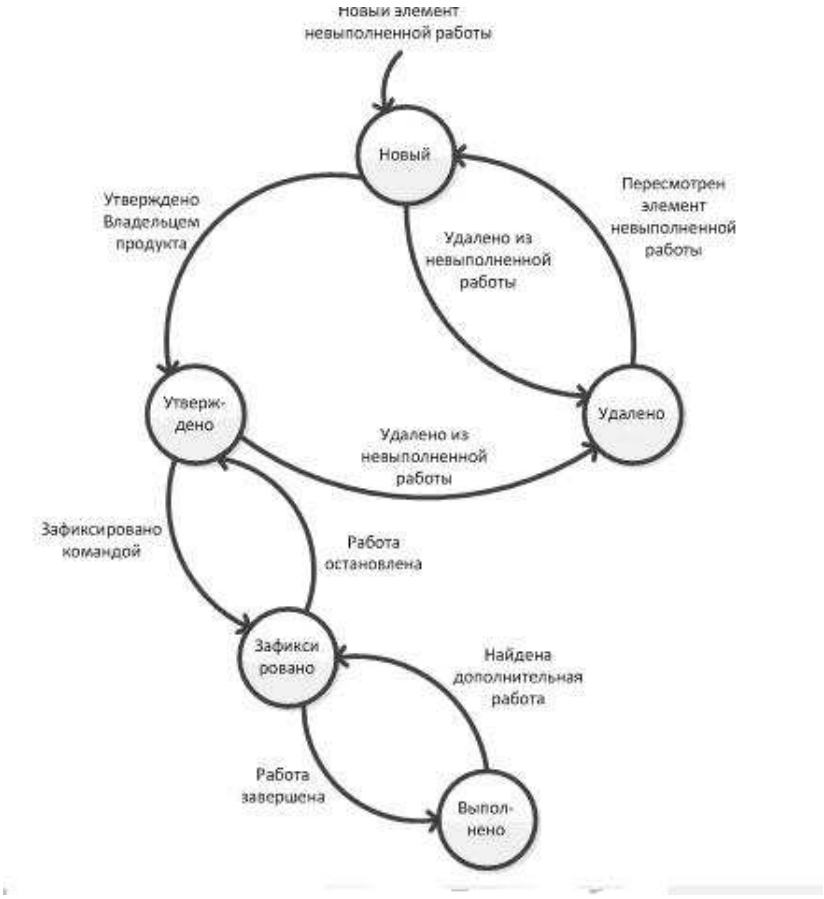

Рис. 8.3. Рабочий процесс элемента невыполненной работы

После установки элементу работы приоритета его состояние изменяют на "Утверждено". На собрании по планированию спринта *команда* просматривает наиболее приоритетные элементы работы и выбирает те, которые будут выполняться в текущем спринте. Для элементов невыполненной работы, которые попали в текущий спринт, устанавливается состояние "Зафиксировано". Это означает тот факт, что рабочие элементы спринта не подлежат изменению до конца спринта. При завершении работы по элементу его состояние устанавливается "Выполнено". Если для элемента работы, находящегося в состоянии "Выполнено", выявляется дополнительная работа, то этот элемент может быть переведен в состояние "Зафиксировано". Для элемента работы, находящегося в состоянии "Зафиксировано", при возникновении проблем, препятствующих его завершению в спринте, может быть работа приостановлена и установлено состояние "Утверждено". Элемент невыполненной работы может быть удален из списка "*Невыполненная работа по продукту*" по решению Владельца продукта. Это может произойти как из состояния "Новый", так и состояния "Утверждено", что соответствует установке для элемента работы состояния "Удалено". В результате пересмотра элемента работы, имеющего состояние "Удалено", для него вновь возможен перевод в состояние "Новый".

Список "*Невыполненная работа по продукту*" является главным документом для Scrum-команды. На основе данного списка *команда* создает другие рабочие элементы, составляющие спринты и выпуски. Для *Элементов невыполненной работы команда* создает задачи и тестовые случаи. Задачи детализируют элемент работы и определяют конкретную реализацию требований пользователя. Тестовые случаи необходимы для проверки соответствия функциональности кода требованиям пользователя. Если тестовый случай не проходит, то создается рабочий элемент "Ошибка". При блокировании задачи из-за невозможности её выполнения в теку-

щем спринте создают рабочий элемент "Препятствие". Scrum-*команда* может создавать вспомогательные рабочие элементы (ошибки и препятствия) в отношении элементов, на которые они влияют (задачи и тестовые случаи) и связывать эти элементы (рис. 8.4).

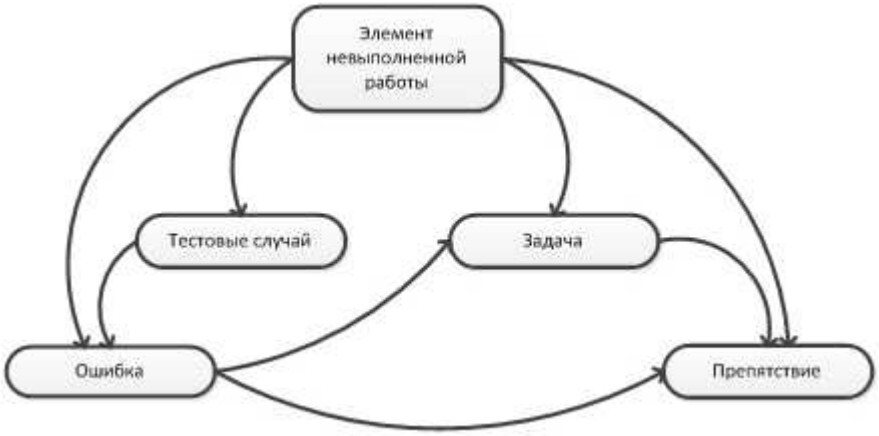

Рис. 8.4. Связь между рабочими элементами

Для отслеживания хода выполнения проекта, можно создавать отчеты, отражающие наиболее важные данные для текущего проекта. В процессе создания *ПО* можно пользоваться стандартными отчетами или создавать собственные отчеты. Отчеты можно создавать, настраивать и просматривать с помощью *Excel*, *Project* или служб Reporting ServicesSQL *Server*.

Методология Scrum имеет следующие положительные стороны:

- пользователи начинают видеть систему спустя всего несколько недель и могут выявлять проблемы на ранних стадиях разработки программного продукта;
- интеграция технических компонентов происходит в ходе каждого спринта и поэтому проблемы проекта (если они возникают) выявляются практически сразу;
- в каждом спринте команда фокусируется на контроле качества;
- гибкая работа с изменениями в проекте на уровне спринта.

Вопросы

1. Поясните основные положения методологии Scrum.
2. Какие артефакты характерны для методологии Scrum?
3. Какие рабочие элементы определены в шаблоне MicrosoftVisualStudioScrum 2.2?
4. Поясните назначение Элементов задела работы продукта.
5. Что представляет собой спринт в методологии Scrum?
6. Какие роли определены в организации команды в методологии Scrum?
7. Кто отвечает за качественный выпуск программного продукта в методологии Scrum?
8. Поясните содержание жизненного цикл проекта ПО в методологии Scrum.
9. Поясните содержание рабочего процесса элемента невыполненной работы.
10. Поясните возможные связи между рабочими элементами.

Библиографический список

1. Орлов С.А. Технологии разработки программного обеспечения. Учебное пособие. – СПб.: Питер, 2003.
2. Андреев Д.В. Организация процессов разработки программного обеспечения с использованием Team Foundation Server 2010 [Электронный ресурс] - URL: https://www.intuit.ru/studies/courses/649/505/info
3. Госсе М., Келлер Б., Кришнамурти А. Вудворт М. Управление жизненным циклом приложений с VisualStudio 2010. Профессиональный подход. – М. ЭКОМ Паблишерз, 2012.
4. Мартин М., Мартин Р. Принципы, паттерны и методики гибкой разработки на языке C#. – СПб.: Символ-Плюс, 2011.
5. Левинсон Дж. Тестирование ПО с помощью Visual Studio 2010. – М.: ЭКОМПаблишерз, 2012.
6. Мейер Дж. Д. Командная разработка с использованием Visual Studio Team Foundation Server / Дж. Д.Мейер, Дж. Тейлор, А. Макман, П. Бансод, К. Джонс – Изд. Корпорация Microsoft, 2007.
7. Отслеживание работы с помощью Visual Studio Online или Team Foundation Server [Электронный ресурс] - URL: https://msdn.microsoft.com/ru-ru/library/dd286619.aspx

Электронное учебное издание

Ольга Викторовна **Свиридова**
Александр Александрович **Рыбанов**
Лидия Александровна **Макушкина**

Технология командной разработки программных систем

Учебно-методическое пособие

Электронное издание сетевого распространения

Редактор Матвеева Н.И.

Темплан 2019 г. Поз. № 1.
Подписано к использованию 11.01.2019. Формат 60x84 1/16.
Гарнитура Times. Усл. печ. л. 4,38.

Волгоградский государственный технический университет.
400005, г. Волгоград, пр. Ленина, 28, корп. 1.

ВПИ (филиал) ВолгГТУ.
404121, г. Волжский, ул. Энгельса, 42а.