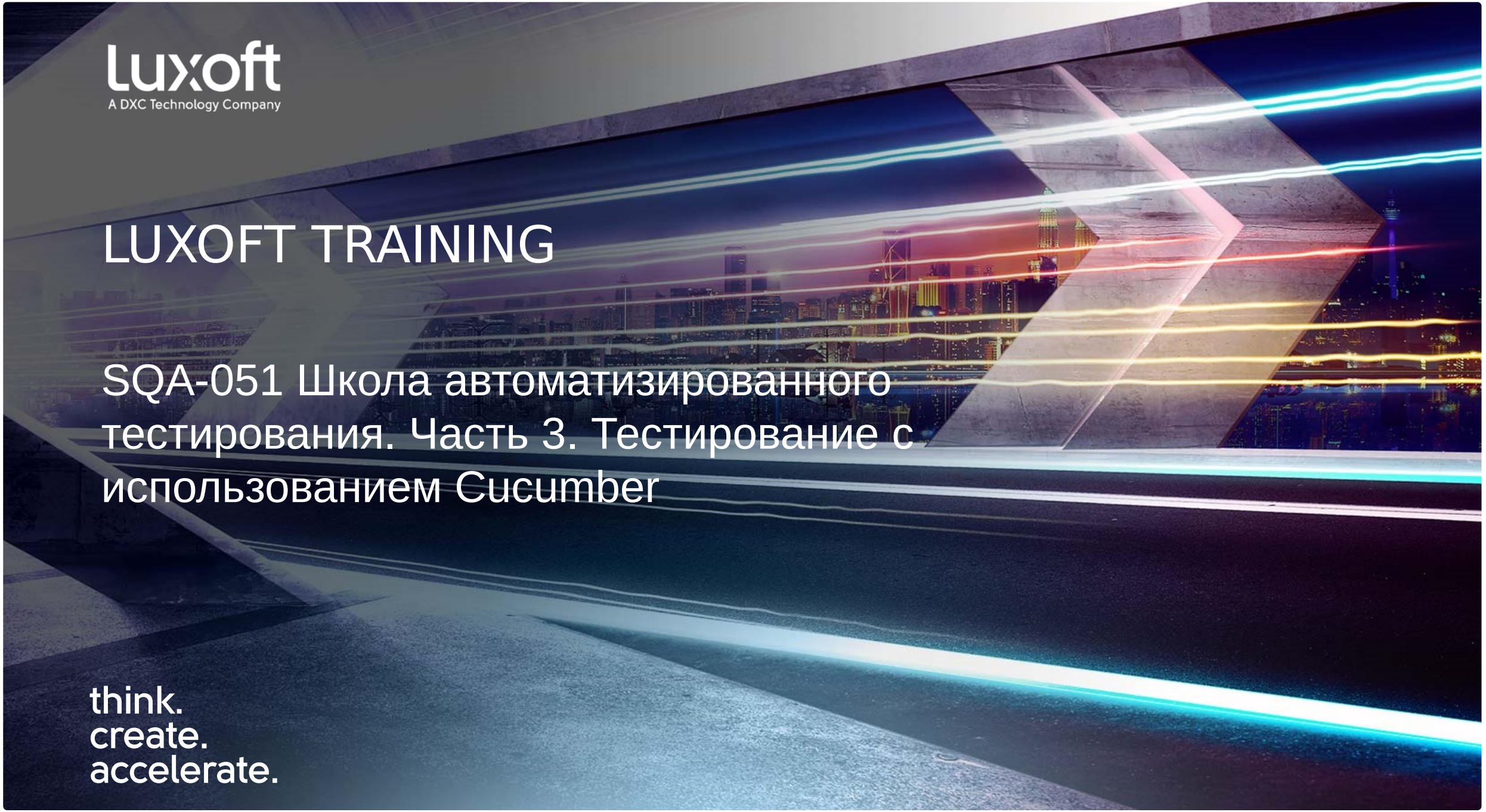

Luxoft
A DXC Technology Company
LUXOFT TRAINING
SQA-051 Школа автоматизированного тестирования. Часть 3. Тестирование с использованием Cucumber
think.
create.
accelerate.

SQA-050 Школа автоматизированного тестирования. Часть 3. Тестирование с использованием Cucumber

Методологии тестирования

СОДЕРЖАНИЕ ТРЕНИНГА

- Методологии разработки 4
- Методологии тестирования 19
- Методология BDD 27

СЕКЦИЯ 1:
МЕТОДОЛОГИИ РАЗРАБОТКИ

ОПРЕДЕЛЕНИЕ

- Модель устоявшихся принципов
- О порядке и способах взаимодействия участников разработки
- Способах принятия решений
- Этапности и порядке приемки проекта

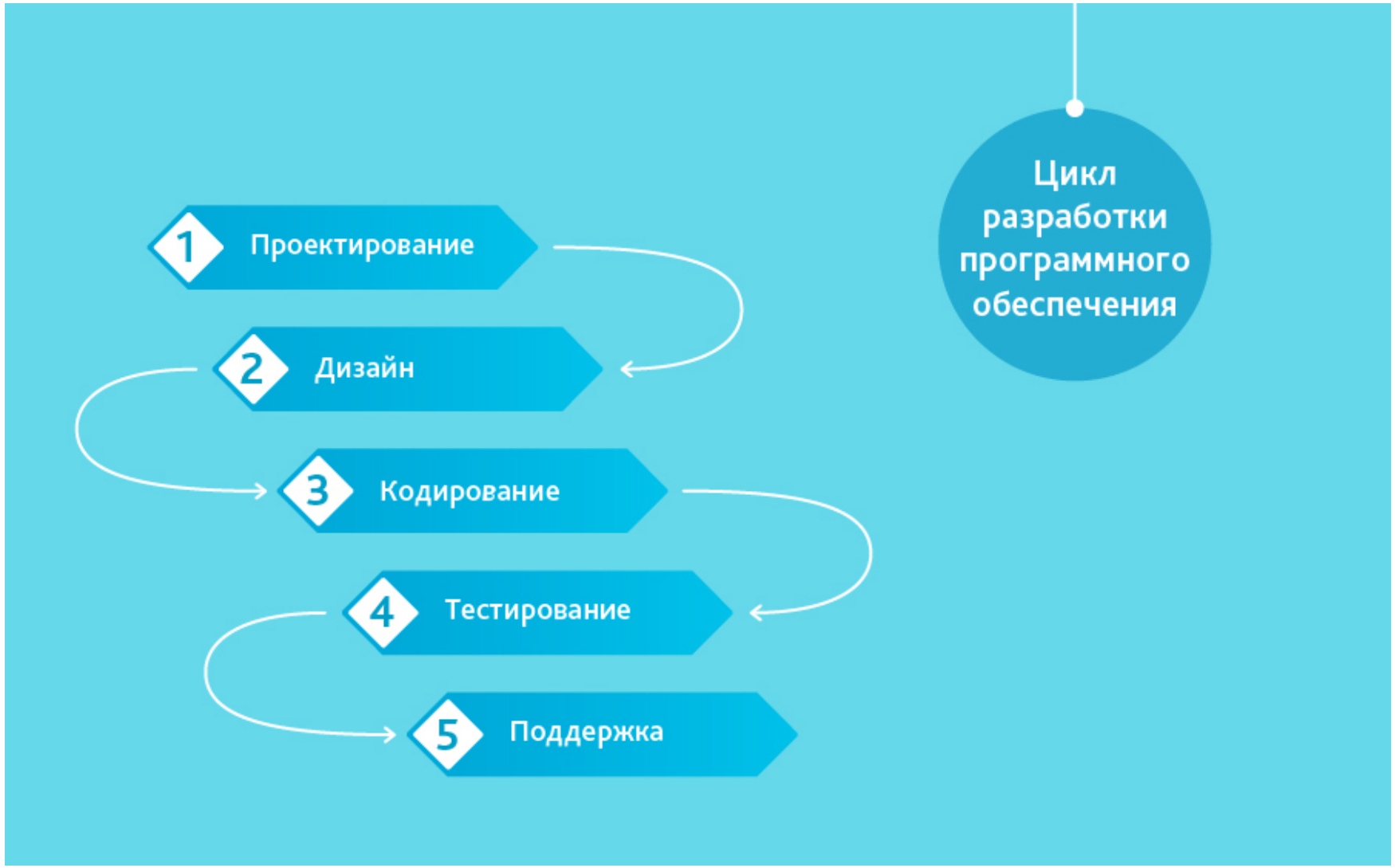
ВИДЫ МОДЕЛЕЙ РАЗРАБОТКИ

- Тяжеловесные (стандартные, «традиционные»)
  - Водопадная
  - V-модель
  - Инкрементальная
  - RAD-модель
  - Итерационная
  - Спиральная
- Легковесные (Agile)
  - Scrum
  - Kanban

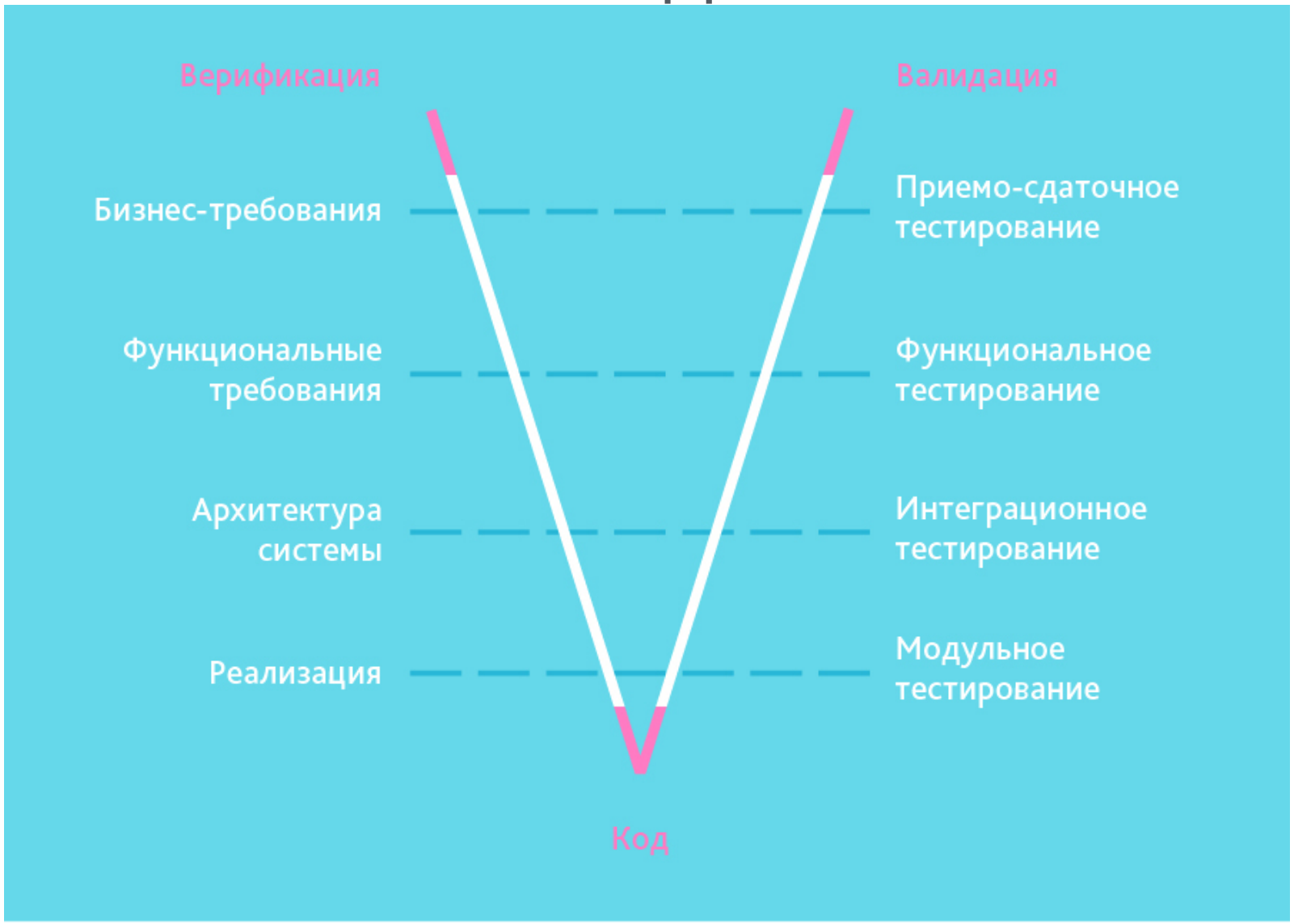
ОСОБЕННОСТИ ТЯЖЕЛОВЕСНОЙ РАЗРАБОТКИ

- Заранее оговоренный список требований, либо порядок их уточнения
- Четко построенный план без существенных изменений
- Обязательность документальной базы для каждого этапа

ВОДОПАДНАЯ


Цикл разработки программного обеспечения
1 Проектирование
2 Дизайн
3 Кодирование
4 Тестирование
5 Поддержка

V-МОДЕЛЬ

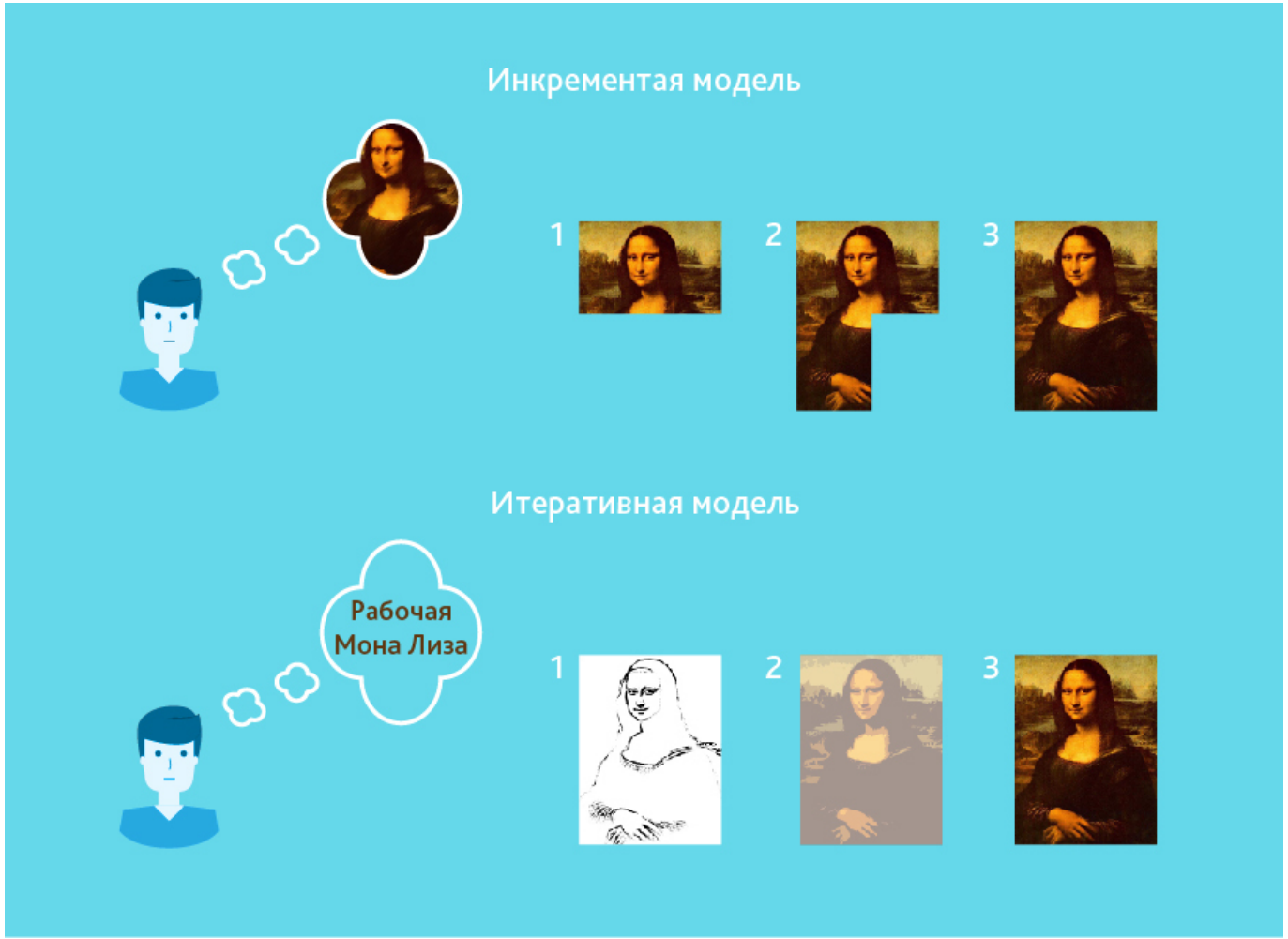

Верификация
Валидация
Бизнес-требования
Приемо-сдаточное тестирование
Функциональные требования
Функциональное тестирование
Архитектура системы
Интеграционное тестирование
Реализация
Модульное тестирование
Код

ИНКРЕМЕНТАЛЬНАЯ

RAD-МОДЕЛЬ


Анализ и быстрое проектирование
Моделирование данных
Моделирование процесса
Сборка приложения
Тестирование
Бизнес-моделирование
Цикл прототипирования команды X
Внедрение

ИТЕРАЦИОННАЯ



СПИРАЛЬНАЯ


Анализ риска (требований)
Формирование требований
1-я версия
2-я версия
3-я версия
Линия принятия решения
Конструирование (проектирование, кодирование, тестирование и интеграция)
Оценка заказчиком. Внедрение и сопровождение

AGILE-МЕТОДОЛОГИИ

- **Люди и взаимодействие** важнее процессов и инструментов.
- **Работающий продукт** важнее исчерпывающей документации.
- **Сотрудничество с клиентом** важнее согласования условий контракта.
- **Готовность к изменениям** важнее следования первоначальному плану.

ОСОБЕННОСТИ AGILE

- Наивысшим приоритетом является удовлетворение потребностей клиента благодаря регулярной и ранней поставке ценного программного обеспечения.
- Изменение требований приветствуется (даже на поздних стадиях разработки).
- Работающий продукт следует выпускать как можно чаще - с периодичностью от пары недель до пары месяцев.
- На протяжении всего проекта разработчики и представители бизнеса должны ежедневно работать вместе.

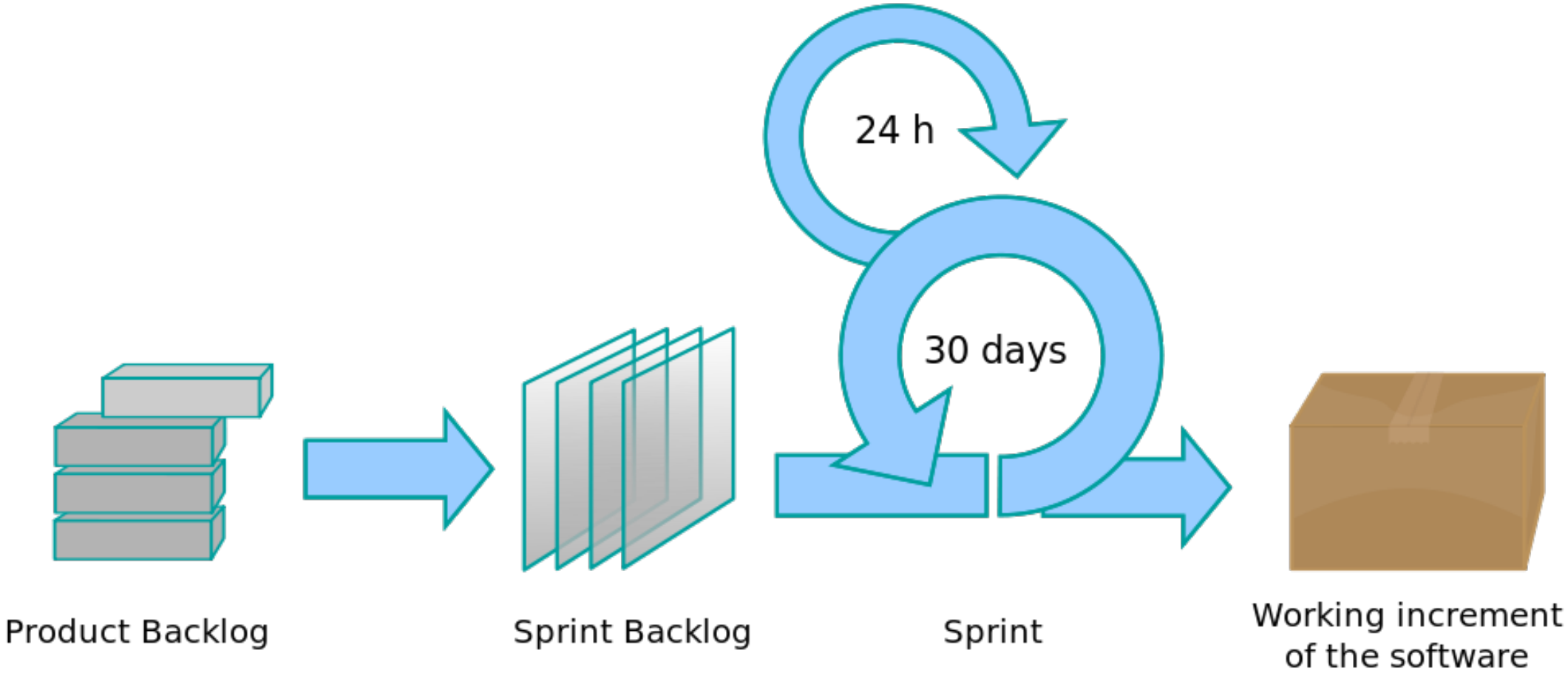
ОСОБЕННОСТИ AGILE

- Над проектом должны работать мотивированные профессионалы. Чтобы работа была сделана, создайте условия, обеспечьте поддержку и полностью доверьтесь им.
- Непосредственное общение является наиболее практичным и эффективным способом обмена информацией как с самой командой, так и внутри команды.
- Работающий продукт — основной показатель прогресса.
- Инвесторы, разработчики и пользователи должны иметь возможность поддерживать постоянный ритм

ОСОБЕННОСТИ AGILE

- Постоянное внимание к техническому совершенству и качеству проектирования повышает гибкость проекта.
- Простота — искусство минимизации лишней работы — крайне необходима.
- Самые лучшие требования, архитектурные и технические решения рождаются у самоорганизующихся команд.
- Команда должна систематически анализировать возможные способы улучшения эффективности и соответственно корректировать стиль своей работы

SCRUM


24 h
30 days
Product Backlog
Sprint Backlog
Sprint
Working increment
of the software

СЕКЦИЯ 2:
МЕТОДОЛОГИИ ТЕСТИРОВАНИЯ

ОПРЕДЕЛЕНИЕ

- Модель устоявшихся принципов
- О порядке и способах взаимодействия участников тестирования
- Способах принятия решений
- Этапности и порядке действий в процессе тестирования

- * т.е. очень похоже на методологии разработки

ЭТАПЫ ТЕСТИРОВАНИЯ

1. Анализ продукта
2. Работа с требованиями
3. Разработка стратегии тестирования и планирование процедур контроля качества
4. Создание тестовой документации
5. Тестирование прототипа
6. Основное тестирование
7. Стабилизация
8. Эксплуатация

АВТОМАТИЗАЦИЯ ТЕСТИРОВАНИЯ: ПИРАМИДА

АВТОМАТИЗАЦИЯ ТЕСТИРОВАНИЯ

Плюсы

- Проверка состояния
- Быстрое выполнение и отчет
- Экономия человеческого ресурса
- Возможность участия разработчиков

Минусы

- Ложная уверенность в качестве
- Ненадежность
- Автоматизация «не равно» тестированию
- Сложность обслуживания
- Медленная разработка новых тестов

СТЕК АВТОМАТИЗАЦИИ: СРЕДСТВА

- Определения логики тестового сценария
- Взаимодействия с тестируемым объектом
- Получения тестовых данных

- Средства объединения всех пунктов выше

СРЕДСТВА ОПРЕДЕЛЕНИЯ ЛОГИКИ ТЕСТОВОГО СЦЕНАРИЯ

- Фреймворк Junit (частично) - фреймворк модульного тестирования программного обеспечения.
- Cucumber- библиотека описания тестовых сценариев на языке, близком к естественному.
- Selenium IDE- среда разработки и запуска автотестов.

СРЕДСТВА ВЗАИМОДЕЙСТВИЯ С ТЕСТИРУЕМЫМ ОБЪЕКТОМ

- Selenium framework - фреймворк взаимодействия с программным обеспечением путем имитации действий пользователя.

- Test complete - альтернативный фреймворк взаимодействия с ПО.

СЕКЦИЯ 3:
МЕТОДОЛОГИЯ BDD

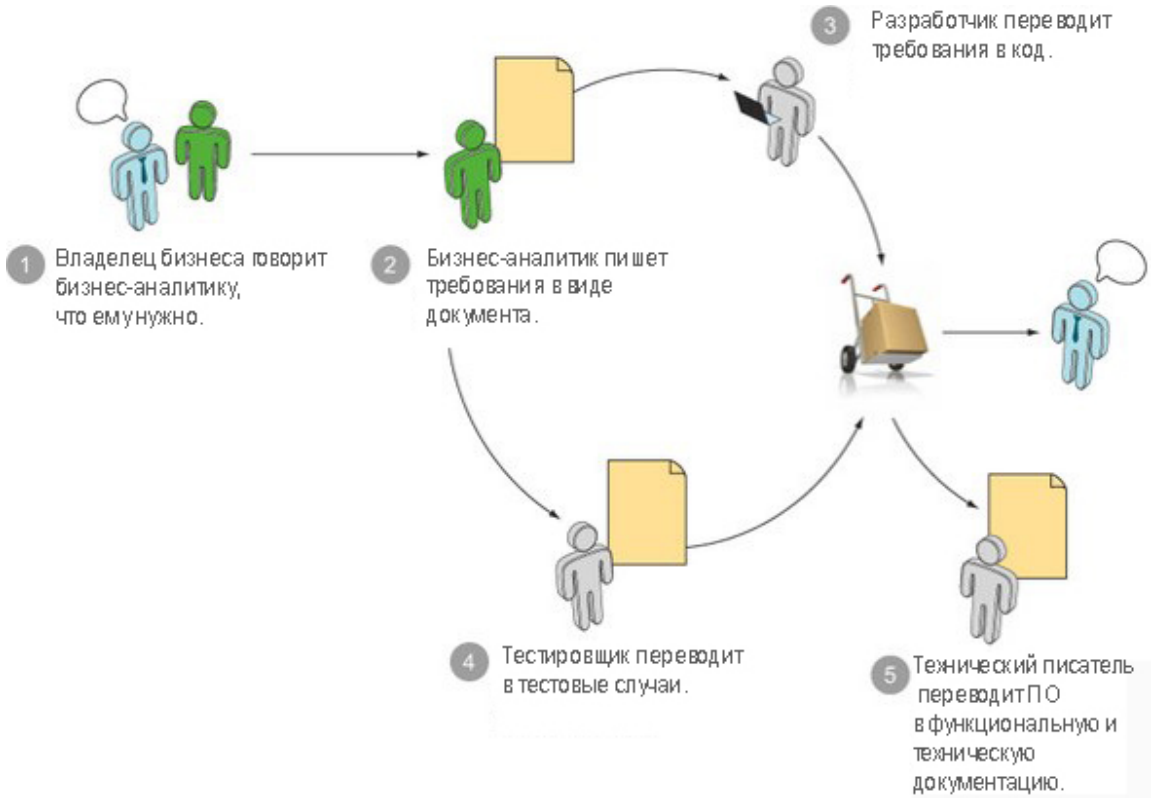
ВВЕДЕНИЕ В BDD

В этом модуле рассматриваются следующие темы:

- Проблемы, решаемые с помощью методологии разработки через тестирование поведения (Behavior-Driven Development – BDD)
- Возникновение и общие принципы разработки через тестирование поведения
- Задачи и результаты в проекте с использованием методологии BDD
- Преимущества и недостатки разработки через

ТРАДИЦИОННЫЙ ПРОЦЕСС РАЗРАБОТКИ

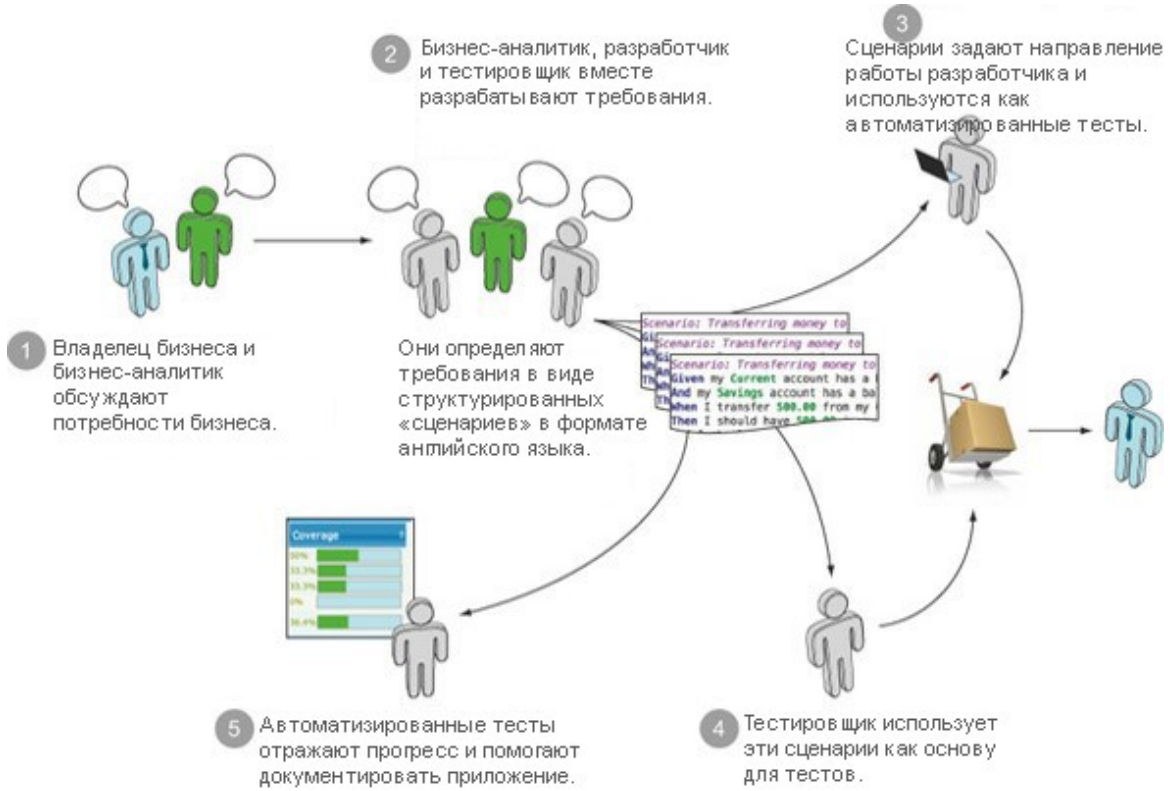
В традиционном процессе разработки возникает множество возможностей для недопонимания и несогласованности действий



ТРАДИЦИОННЫЙ ПРОЦЕСС РАЗРАБОТКИ

ПРОЦЕСС РАЗРАБОТКИ ЧЕРЕЗ ТЕСТИРОВАНИЕ ПОВЕДЕНИЯ (BDD)

В процессе BDD используются диалоги относительно примеров в легко автоматизируемой форме, чтобы сократить потери информации и недопонимание

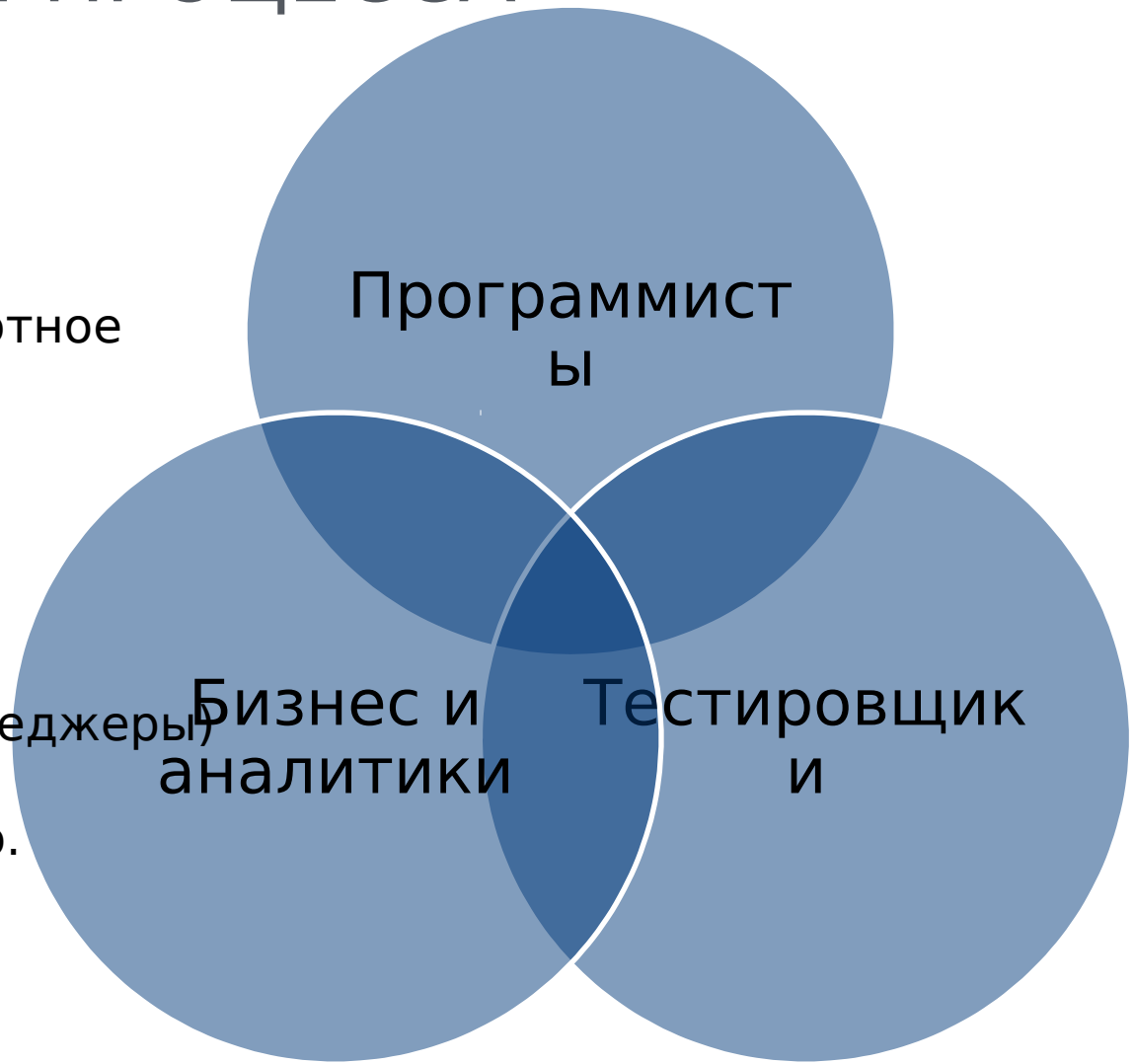


УЧАСТНИКИ ПРОЦЕССА

Процесс тестирования BDD предполагает плотное сотрудничество таких членов команды как:

- Разработчики ПО
- Представители бизнеса и аналитики
- Тестировщики (автоматизаторы и тест-менеджеры)

Каждая команда получает свои бонусы за это.

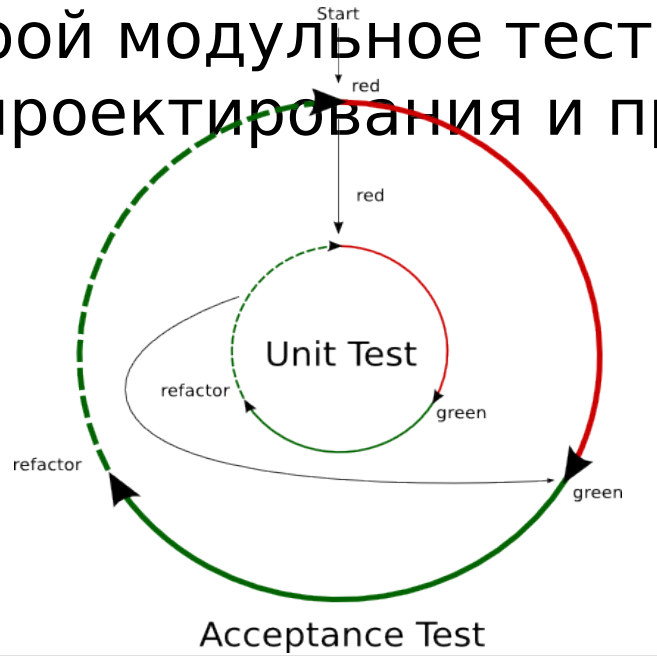


ВВЕДЕНИЕ В ПРОЦЕСС РАЗРАБОТКИ ЧЕРЕЗ ТЕСТИРОВАНИЕ ПОВЕДЕНИЯ

- Разработка через тестирование поведения (BDD) – это набор инженерных практик, призванных помочь командам в создании более ценного и качественного ПО в кратчайшие сроки.
- Этот процесс основан на методологии Agile и практиках бережливой разработки, в частности таких как разработка через тестирование (Test-Driven Development – TDD) и предметно-ориентированное проектирование (Domain-Driven Design – DDD).
- Главное – BDD дает участникам разработки единый язык, основанный на простых, структурированных предложениях на английском (или на родном языке заинтересованных лиц), который существенно упрощает коммуникацию между членами проектных команда и бизнес-стейкхолдерами.

ВВЕДЕНИЕ В ПРОЦЕСС РАЗРАБОТКИ ЧЕРЕЗ ТЕСТИРОВАНИЕ ПОВЕДЕНИЯ

- Процесс BDD изначально был придуман Дэном Нортом в начале 2000-х годов как простой способ практического обучения разработке через тестирование (TDD).
- Процесс TDD, предложенный Кентом Беком вскоре после появления методологии Agile, – это замечательная эффективная методология, в которой модульное тестирование используется для спецификации, проектирования и проверки прикладного кода.



РАЗРАБОТКА ЧЕРЕЗ ТЕСТИРОВАНИЕ

Разработка через тестирование основывается на простом, трехфазном цикле

РАЗРАБОТКА ЧЕРЕЗ ТЕСТИРОВАНИЕ

- Несмотря на преимущества этого метода, многие команды испытывают трудности при переходе к эффективному применению TDD.
- Разработчики не знают, с чего начать или какие тесты нужно написать далее.
- Иногда при использовании TDD разработчики чрезмерно сосредоточиваются на деталях, упуская из виду более общую картину бизнес-целей, которые необходимо реализовать.
- Некоторые команды обнаруживают, что им становится трудно поддерживать большое количество модульных тестов по мере расширения проекта.

РАЗРАБОТКА ЧЕРЕЗ ТЕСТИРОВАНИЕ

- На самом деле, многие традиционные модульные тесты, написанные с применением или без применения TDD, тесно увязаны с конкретной реализацией кода. Они ориентированы на тестируемые методы или функции, а не на задачи, которые программа должны выполнять с точки зрения бизнеса.

РАЗРАБОТКА ЧЕРЕЗ ТЕСТИРОВАНИЕ

```
public class BankAccountTest {
        @Test
        public void testTransfer() {...}

        @Test
        public void testDeposit() {...}
    }
```

РАЗРАБОТКА ЧЕРЕЗ ТЕСТИРОВАНИЕ

- Такие тесты лучше, чем ничего, но они ограничивают наши возможности.
- Они не описывают то, что вы ожидаете от выполнения функций transfer() и deposit(), и поэтому их труднее понять и исправить возможные ошибки.
- Они тесно увязаны с тестируемым методом, а значит при рефакторинге реализации необходимо переименовывать соответствующие тесты.

РАЗРАБОТКА ЧЕРЕЗ ТЕСТИРОВАНИЕ

- Норт заметил, что несколько простых приемов, например именование модульных тестов полными предложениями и использование слова “should”, может помочь разработчикам в написании более осмысленных тестов, что, в свою очередь, поможет им более эффективно писать качественный код.
- Если думать о том, что должен выполнять класс, а не о том, какой метод или функция тестируется, то будет проще направить усилия на реализацию соответствующих бизнес-требований.

РАЗРАБОТКА ЧЕРЕЗ ТЕСТИРОВАНИЕ

```
public class WhenTransferringInternationalFunds {
    @Test
    public void should_transfer_funds_to_a_local_account() {...}


    @Test
    public void should_transfer_funds_to_a_different_bank() {...}


    @Test
    public void should_deduct_fees_as_a_separate_transaction() {...}

}
```

РАЗРАБОТКА ЧЕРЕЗ ТЕСТИРОВАНИЕ ПОВЕДЕНИЯ

Идея

- Совмещение интересов
- Команды разработки
- И команды бизнеса и аналитиков
- Путем включения бизнеса в процесс разработки тестовых сценариев
- С использованием специализированных программных

Варианты реализации

- Аналитики и бизнес пишут тестовые сценарии на естественном языке
- Фреймворк преобразует эти сценарии в программный код
- Который потом дополняется разработчиком

РАЗРАБОТКА ЧЕРЕЗ ТЕСТИРОВАНИЕ РАЗРАБОТКА ЧЕРЕЗТЕСТИРОВАНИЕ ПОВЕДЕНИЯ

- Написанные таким образом тесты читаются скорее как спецификации, чем как модульные тесты.
- Такие тесты ориентированы на поведение приложения; они используются просто как средство описания и проверки этого поведения.
- Норт также заметил, что написанные таким образом тесты гораздо проще поддерживать, поскольку их цель более очевидна.
- Влияние этого подхода было столь велико, что он начал называть этот метод не разработкой через тестирование, а разработкой через тестирование поведения.

BDD ОТЛИЧНО РАБОТАЕТ ДЛЯ АНАЛИЗА ТРЕБОВАНИЙ

- Однако описание поведения системы – это то, чем ежедневно занимаются бизнес-аналитики.
- Работая со своим коллегой, бизнес-аналитиков Крисом Мэттсом, Норт решил применить полученные знания в области анализа требований.
- Примерно тогда же Эрик Эванс выдвинул идею предметно-ориентированного программирования, в котором для описания и моделирования системы используется универсальный язык, понятный представителям бизнеса.
- Норт и Мэттс стремились создать единый язык, который могли бы использовать и бизнес аналитики, чтобы однозначно определять требования, и который можно было бы легко трансформировать в автоматизированные приемочные тесты.
- Для этого они решили выражать критерии приемки для пользовательских историй в форме свободно структурированных

BDD ОТЛИЧНО РАБОТАЕТ ДЛЯ АНАЛИЗА ТРЕБОВАНИЙ

```
Given a customer has a current account
When the customer transfers funds from this account to an overseas account
Then the funds should be deposited in the overseas account
And the transaction fee should be deducted from the current account
```

BDD ОТЛИЧНО РАБОТАЕТ ДЛЯ АНАЛИЗА ТРЕБОВАНИЙ

- Владелец бизнеса может легко понять написанный таким образом сценарий. В нем указаны четкие и объективные цели для каждой истории, что нужно разработать и что нужно протестировать.
- Такое представление со временем развилось в общеупотребительную форму языка, известного под названием Gherkin. С помощью соответствующих инструментов написанные в этой форме сценарии можно превратить в автоматизированные приемочные тесты, которые автоматически выполняются, когда это необходимо.
- Первую специальную библиотеку JBehave для автоматизации тестирования по методологии BDD Дэн Норт написал в середине 2000-х, и затем появилось множество таких библиотек для различных языков, как на уровне модульного тестирования, так и на уровне приемочного тестирования.

ВЫГОДЫ

Бизнес

- Гарантия тестирования важных сценариев
- Четкий контракт с командами разработки/ тестирования
- Инструмент моделирования своих интересов в продукте

Команда тестирования

- Защита от «мутирующих требований»
- Точно сформулированные требования
- Доказательная эффективность своей работы

ПРИНЦИПЫ И ПРАКТИКИ BDD

- Основное внимание на функции, реализующие ценность для бизнеса;
- Совместная работа по определению функций;
- Исключение неопределенности;
- Иллюстрация функций конкретными примерами;
- Не пишите автоматизированные тесты, пишите выполняемые спецификации;
- Не пишите модульные тесты, пишите спецификации низкого уровня;
- Создание актуальной документации
- Использование актуальной документации для поддержки сопровождения

ОСНОВНОЕ ВНИМАНИЕ НА ФУНКЦИИ, РЕАЛИЗУЮЩИЕ ЦЕННОСТЬ ДЛЯ БИЗНЕСА

- Функция – это значимая, поставляемая часть функциональности, необходимая для достижения бизнес-целей.
- Вместо того, чтобы раз и навсегда определить требования, команды, практикующие BDD, ведут постоянный диалог с конечными пользователями и другими заинтересованными лицами, добиваясь общего понимания того, какие функции необходимо реализовать.

СОВМЕСТНАЯ РАБОТА ПО ОПРЕДЕЛЕНИЮ ФУНКЦИЙ

- BDD – это, прежде всего, практика взаимодействия пользователей и команды разработки, а также взаимодействия внутри самой команды. Бизнес-аналитики, разработчики и тестировщики взаимодействуют с конечными пользователями, чтобы определить и конкретизировать функции, а члены команды генерируют идеи, основываясь на своем опыте и знаниях.
- Если попросить пользователей написать, что они хотят получить, то они, как правило, выдадут набор подробных требований, который показывает то, каким они видят решение. Другими словами, пользователи не скажут, что им нужно, в скорее предложат вам

ИСКЛЮЧЕНИЕ НЕОПРЕДЕЛЕННОСТИ

- Команда BDD понимает, что невозможно все знать заранее, сколько бы времени они не потратили на написание спецификаций.
- Вместо того, чтобы зафиксировать спецификации в самом начале проекта, команда, практикующая BDD, исходит из того, что требования (или вернее их понимание требований) будут изменяться по мере реализации проекта.
- Зачастую лучший способ понять, нравится ли пользователям та или иная функция, – реализовать и показать ее им как можно раньше.

ИЛЛЮСТРАЦИЯ ФУНКЦИЙ КОНКРЕТНЫМИ ПРИМЕРАМИ

- Реализуя определенную функцию, команда BDD взаимодействует с пользователями и другими заинтересованными лицами, чтобы уточнить истории и сценарии действий, которые пользователи ожидают от этой функции. В частности, пользователи помогают определить набор конкретных примеров, иллюстрирующих основные действия функции.

ИЛЛЮСТРАЦИЯ ФУНКЦИЙ КОНКРЕТНЫМИ ПРИМЕРАМИ

Реализуя определенную функцию, команда BDD взаимодействует с пользователями и другими заинтересованными лицами, чтобы уточнить истории и сценарии действий, которые пользователи ожидают от этой функции. В частности, пользователи помогают определить набор конкретных примеров, иллюстрирующих основные действия функции.

ИЛЛЮСТРАЦИЯ ФУНКЦИЙ КОНКРЕТНЫМИ ПРИМЕРАМИ

Примеры первичной роли в BDD для более ясного понимания требований

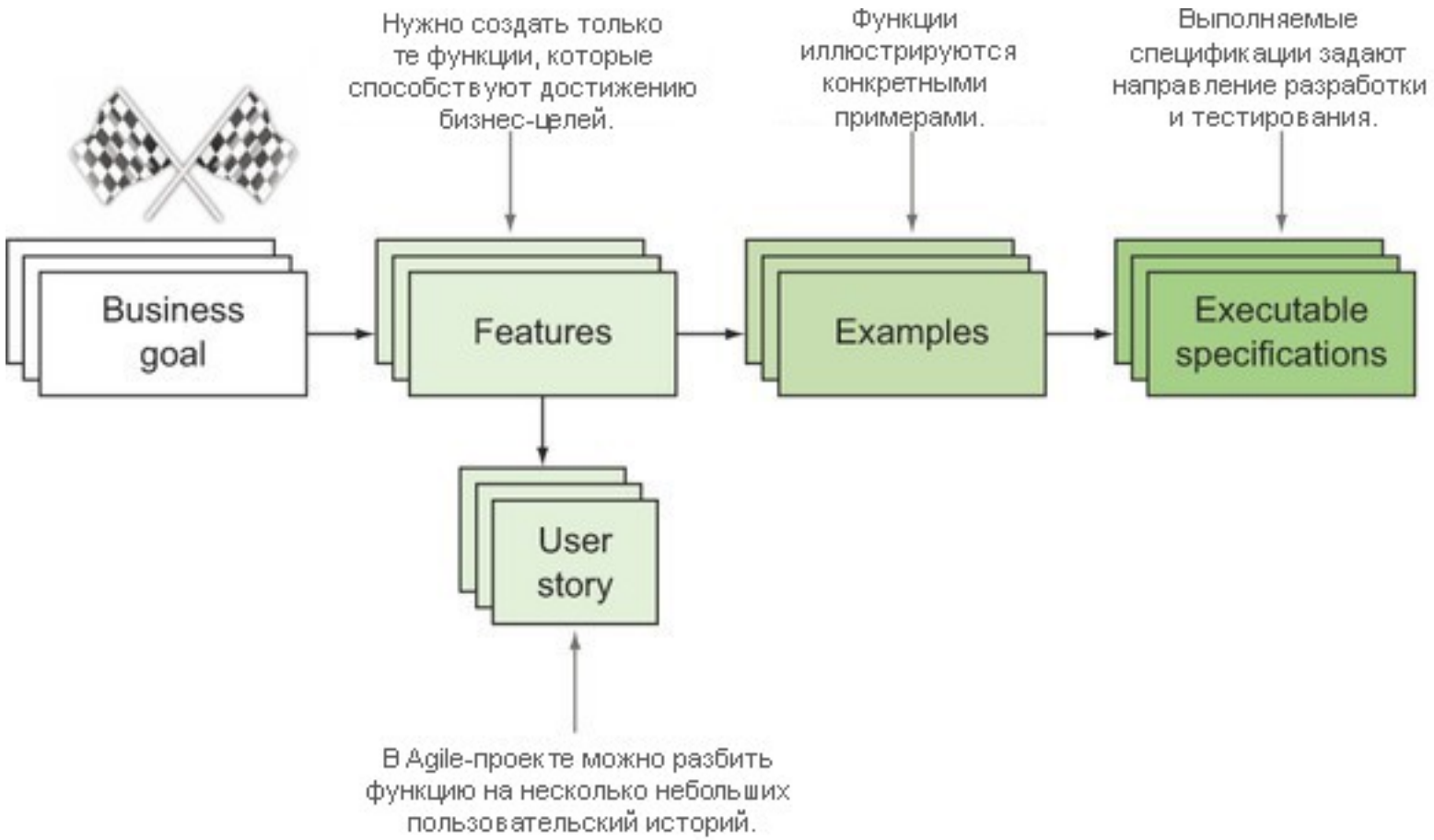


ИЛЛЮСТРАЦИЯ ФУНКЦИЙ КОНКРЕТНЫМИ ПРИМЕРАМИ

- Для этих примеров используется общий словарь, и они легко понятны как конечным пользователям, так и членам команды разработки. Обычно они выражаются с помощью нотаций Given ... When ... Then, как показано на слайде 45.
- На следующем слайде представлен простой пример, который иллюстрирует, как может выглядеть функция “Transfer funds between a client’s accounts” («Перевод средств между счетами клиента»):

ИЛЛЮСТРАЦИЯ ФУНКЦИЙ КОНКРЕТНЫМИ ПРИМЕРАМИ

```
Scenario: Transferring money to a savings account
Given I have a current account with 1000.00
And I have a savings account with 2000.00
When I transfer 500.00 from my current account to my savings account
Then I should have 500.00 in my current account
And I should have 2500.00 in my savings account
```

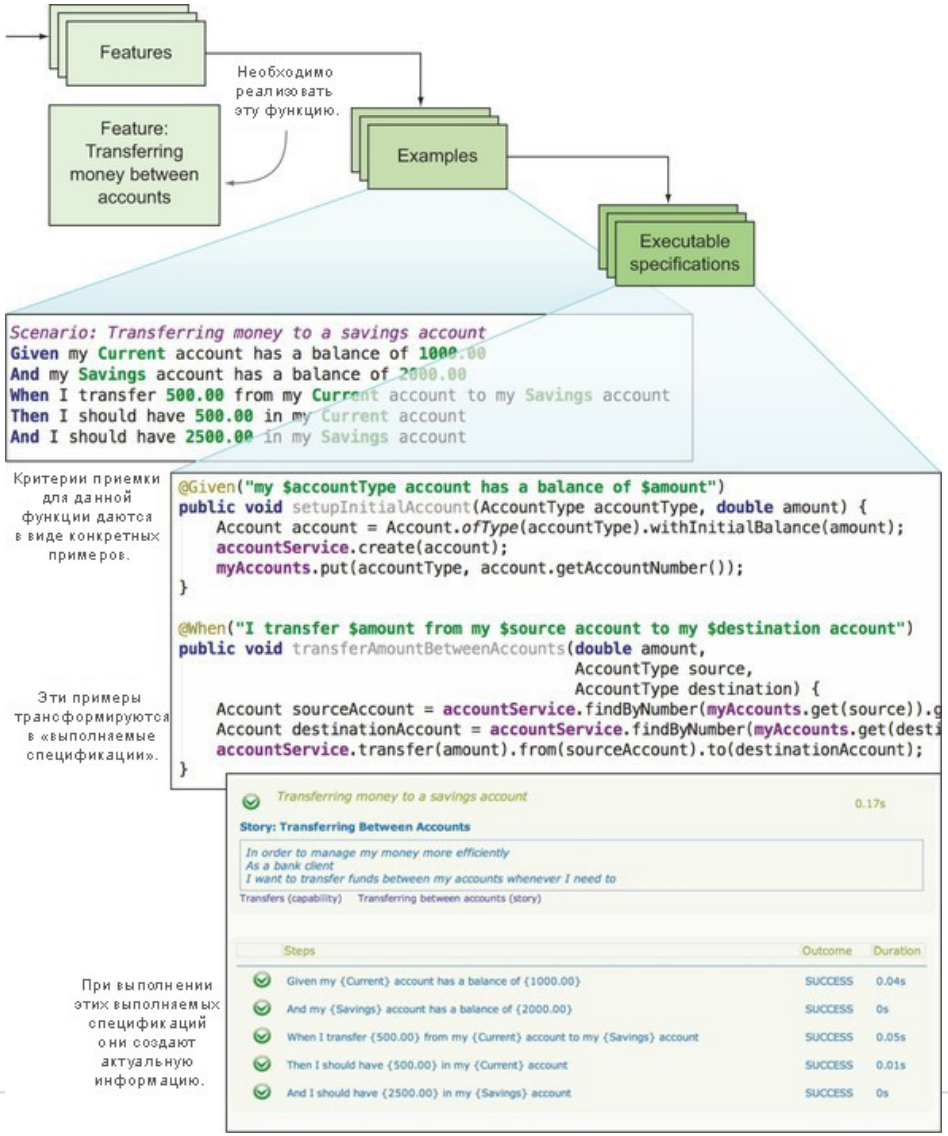
ИЛЛЮСТРАЦИЯ ФУНКЦИЙ КОНКРЕТНЫМИ ПРИМЕРАМИ

- Примеры играют важную роль в BDD просто потому, что они являются чрезвычайно эффективным способом передачи ясных, точных и недвусмысленных требований. Спецификации, написанные на естественном языке, очень плохо выражают требования, поскольку остается простор для двусмысленности, предположений и неправильного понимания. Примеры – отличны способ преодолеть эти ограничения и прояснить требования.
- Кроме того, примеры прекрасно помогают углублять и расширять знания. Когда пользователь предлагает пример того, как должна действовать функция, члены проектной команды обычно просят привести еще

НЕ ПИШИТЕ АВТОМАТИЗИРОВАННЫЕ ТЕСТЫ, ПИШИТЕ ВЫПОЛНЯЕМЫЕ СПЕЦИФИКАЦИИ

- Эти истории и примеры становятся основой спецификаций, используемых разработчиками для создания системы. Они служат одновременно и критериями приемки, определяющими готовность функции, и руководством для разработчиков, дающим им ясное представление о том, что необходимо создать.
- Критерии приемки позволяют команде разработки объективно оценить, насколько правильно реализована та или иная функция. Проверка вручную каждого изменения кода, как правило, неэффективна и отнимает много времени. Такие проверки замедляют получение обратной связи, что, в свою очередь, замедляет процесс разработки. Там, где это возможно, команды трансформируют эти критерии приемки в автоматизированные приемочные тесты или, точнее, в выполняемые спецификации.
- Выполняемая спецификация – это автоматизированный тест, который иллюстрирует и проверяет то, как приложение выполняет конкретное бизнес-требование. Такие автоматизированные тесты выполняются, когда в приложение вносятся какие-либо изменения. Они служат как приемочными тестами, определяя, какие новые функции готовы, и регрессионными тестами, подтверждающим, что внесенные изменения не нарушили существующие функции.

НЕ ПИШИТЕ АВТОМАТИЗИРОВАННЫЕ ТЕСТЫ, ПИШИТЕ ВЫПОЛНЯЕМЫЕ СПЕЦИФИКАЦИИ



НЕ ПИШИТЕ АВТОМАТИЗИРОВАННЫЕ ТЕСТЫ, ПИШИТЕ ВЫПОЛНЯЕМЫЕ СПЕЦИФИКАЦИИ

- В отличие от обычных модульных и интеграционных тестов или автоматизированных функциональных тестов, к которым привыкли многие тестировщики, выполняемые спецификации написаны на языке, близком к естественному. В ним используются именно те примеры, которые ранее предложили и уточнили пользователи и члены команды разработки, используя те же самые условия и словарь. Выполняемые спецификации обеспечивают не только проверку функций, но и взаимодействие, а создаваемые отчеты понятные всем участникам проекта.
- Кроме того, выполняемые спецификации становятся единым источником достоверных данных, эталонным описанием того, как должны быть реализованы функции. Благодаря этому намного проще поддерживать требования. Если спецификации хранятся в виде документа Word или страницы Wiki page, как это делается во многих традиционных проектах, то любые изменения требований должны быть отражены как самом документе с требованиями, так и в приемочных тестах и сценариях тестирования, что влечет за собой риск возникновения несогласованности. Для команд, практикующих BDD, требования и выполняемые спецификации – это одно и то же: когда изменяются требования, выполняемые спецификации сразу

НЕ ПИШИТЕ МОДУЛЬНЫЕ ТЕСТЫ, ПИШИТЕ СПЕЦИФИКАЦИИ НИЗКОГО УРОВНЯ

- BDD не ограничивается приемочными тестами. Методология BDD помогает разработчиками создавать качественный код, более надежный, более удобный в сопровождении и лучше документированный.
- Прежде чем приступить к написанию кода, BDD разработчик выясняет, какие именно функции должна выполнять программа, и выражает их в виде низкоуровневых выполняемых спецификаций.
- Эти низкоуровневые спецификации, естественным образом вытекающие из высокоуровневых критериев приемки, помогают разработчикам в написании и документировании кода приложения в процессе реализации высокоуровневых функций.

НЕ ПИШИТЕ МОДУЛЬНЫЕ ТЕСТЫ, ПИШИТЕ СПЕЦИФИКАЦИИ НИЗКОГО УРОВНЯ

Высокоуровневые критерии приемки в виде выполняемых спецификаций.

```
Scenario: Transferring money to a savings account
Given my Current account has a balance of 1000.00
And my Savings account has a balance of 2000.00
When I transfer 500.00 from my Current account to my Savings account
Then I should have 500.00 in my Current account
And I should have 2500.00 in my Savings account
```

Определения шагов запускают код для реализации шагов в критериях приемки.

```
@Given("my $accountType account has a balance of $amount")
public void setupInitialAccount(AccountType accountType, double amount) {
    Account account = Account.ofType(accountType).withInitialBalance(amount);
    accountService.create(account);
    myAccounts.put(accountType, account.getAccountNumber());
}

@When("I transfer $amount from my $source account to my $destination account")
public void transferAmountBetweenAccounts(double amount,
                                          AccountType source,
                                          AccountType destination) {
    Account sourceAccount = accountService.findByNumber(myAccounts.get(source)).g
    Account destinationAccount = accountService.findByNumber(
    accountService.transfer(amount).from(so
```

Низкоуровневые выполняемые спецификации (модульные тесты) помогают проектировать детальную реализацию.

```
class WhenCreatingANewAccount extends Specification {

    def "account should have a number, a type and an initial balance"() {
        when:
            Account account = Account.ofType(Savings)
                                     .withInitialBalance(100)
        then:
            account.accountType == Savings
            account.balance == 100
    }
```

Luxoft | training
A DXC Technology Company

НЕ ПИШИТЕ МОДУЛЬНЫЕ ТЕСТЫ, ПИШИТЕ СПЕЦИФИКАЦИИ НИЗКОГО УРОВНЯ

```
@Given("my $accountType account has a balance of $amount")
public void setupInitialAccount(AccountType type, double amount) {
    Account account = Account.ofType(type)
                             .withInitialBalance(amount);
    ...
}
```

Create a new account of given type and with given initial balance.

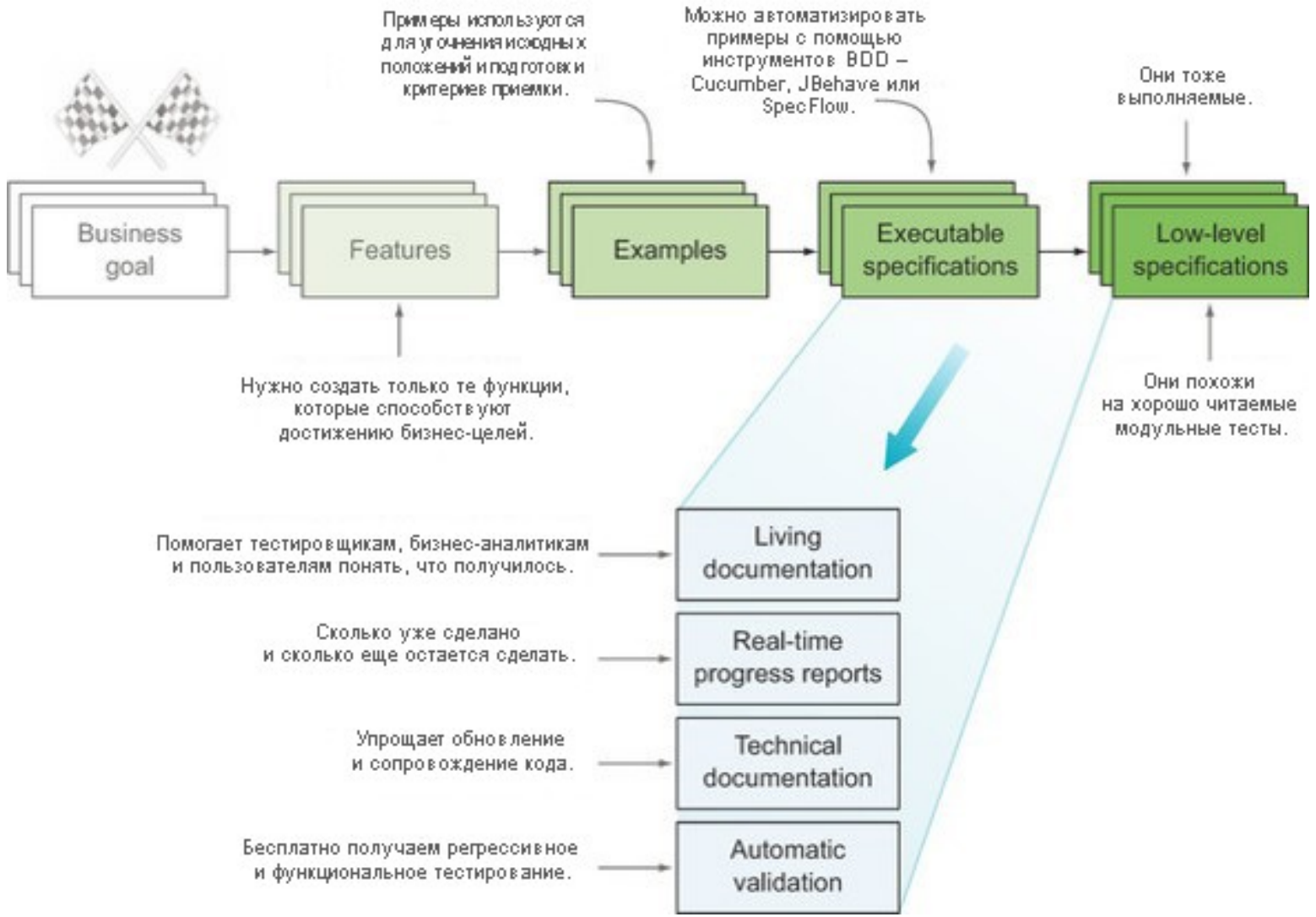
```
class WhenCreatingANewAccount extends Specification {


    def "account should have a type and an initial balance" () {
        when:
        Account account = Account.ofType(Savings)
                .withInitialBalance(100)
        then:
        account.accountType == Savings
        account.balance == 100
    }
}
```

СОЗДАНИЕ АКТУАЛЬНОЙ ДОКУМЕНТАЦИИ

- Создаваемые выполняемыми спецификациями отчеты – это не просто технические отчеты для разработчиков, но эффективная форма документации продукта для всей команды, причем на языке знакомом и понятном пользователям (см. следующий слайд). Эта документация всегда актуальна и не требует особых усилий для поддержания или обновления вручную. Она автоматически создается из последней версии приложения. Каждая функция приложения описывается в понятных терминах и иллюстрируется несколькими ключевыми примерами. Для веб-приложений такого рода актуальная документация обычно включает в себя скриншоты приложения для каждой функции.

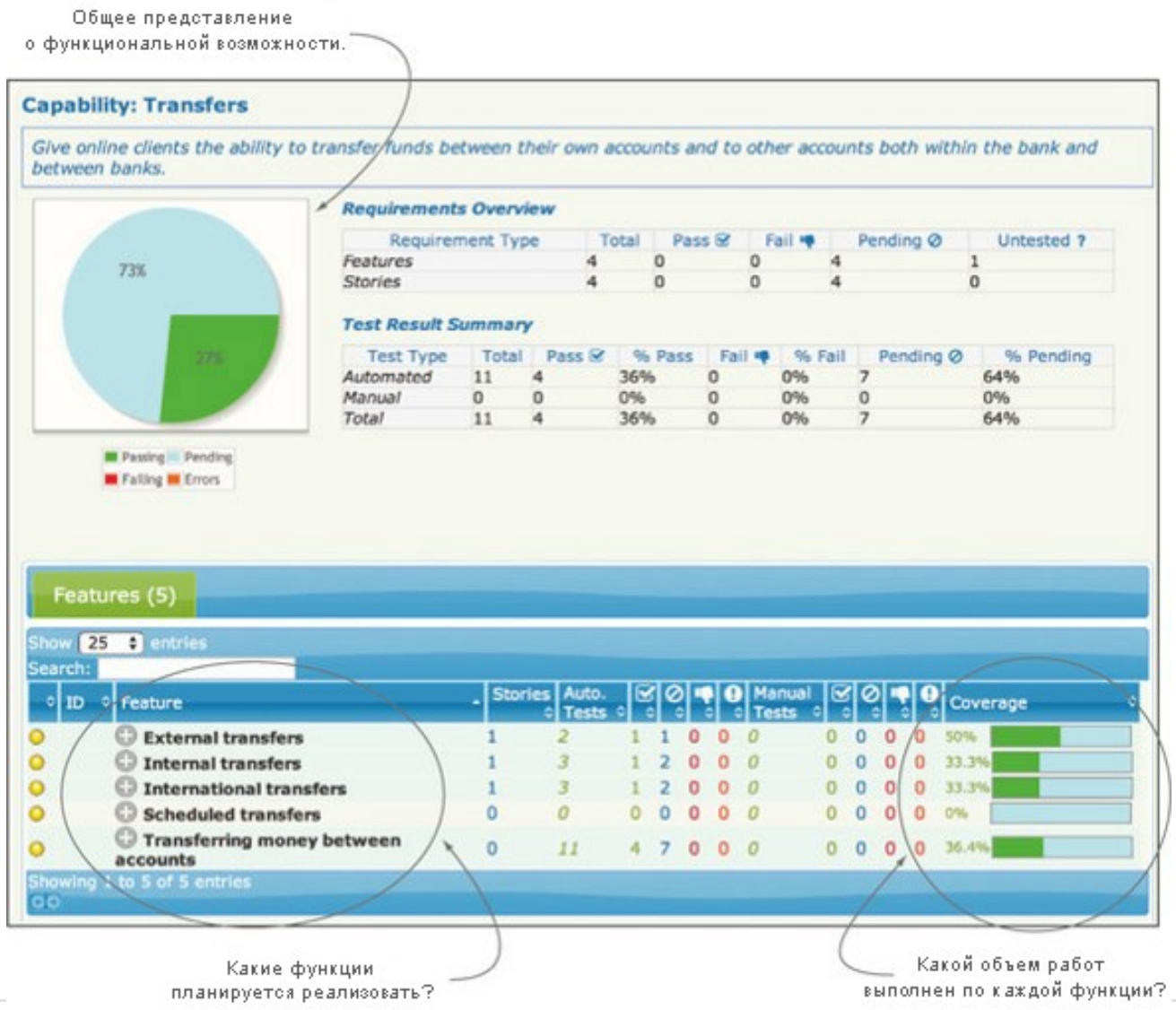
СОЗДАНИЕ АКТУАЛЬНОЙ ДОКУМЕНТАЦИИ



СОЗДАНИЕ АКТУАЛЬНОЙ ДОКУМЕНТАЦИИ

- Опытные команды организуют документацию таким образом, чтобы она легко читалась и ее было удобно использовать всеми участниками проекта (см. рисунок на следующем слайде). Разработчики могут обращаться к документации, чтобы посмотреть, как работают существующие функции. Тестировщики и бизнес-аналитики могут посмотреть, как реализованы интересующих их функции. Владельцы продукта и менеджеры проекта могут использовать режим общего просмотра, чтобы оценить текущее состояние проекта, посмотреть прогресс и решить, как функции можно запустить в производство. Пользователи могут посмотреть в документации, какие возможности есть у приложения и как оно работает.

СОЗДАНИЕ АКТУАЛЬНОЙ ДОКУМЕНТАЦИИ



ИСПОЛЬЗОВАНИЕ АКТУАЛЬНОЙ ДОКУМЕНТАЦИИ ДЛЯ ПОДДЕРЖКИ СОПРОВОЖДЕНИЯ

- Преимущества актуальной документации и выполняемых спецификаций очевидны и после завершения проекта. Систему, разработанную с использованием этих практик, значительно проще и дешевле для сопровождения.
- По словам Роберта Гласса (который ссылается на другие источники), стоимость сопровождения ПО составляет от 40% до 80% от стоимости его разработки. Хотя многие команды отмечают, что количество дефектов существенно сокращается при использовании методологии BDD, дефекты все равно случаются. Постоянное улучшение приложения – естественная части процесса его разработки и использования.

ИСПОЛЬЗОВАНИЕ АКТУАЛЬНОЙ ДОКУМЕНТАЦИИ ДЛЯ ПОДДЕРЖКИ СОПРОВОЖДЕНИЯ

- Во многих организациях, когда проект идет в производство, он передается другой команде для сопровождения. Разработчики, участвующие в таком сопровождении, обычно не принимали участия в разработке проекта, и им необходимо изучить код с нуля. Понятная и актуальная техническая документация существенно облегчает эту задачу.

ИСПОЛЬЗОВАНИЕ АКТУАЛЬНОЙ ДОКУМЕНТАЦИИ ДЛЯ ПОДДЕРЖКИ СОПРОВОЖДЕНИЯ

Кроме того, проще отследить влияние вызванных сопровождением изменений на существующий код. Когда разработчик вносит изменение, оно может привести к прерыванию работы выполняемых спецификаций, и это может произойти по двум причинам:

- Такая выполняемая спецификация больше не отражает новые бизнес-требования. В этом случае выполняемая спецификация может быть обновлена или удалена (если она больше не нужна).
- Изменение кода привело к нарушению существующего требования. Это ошибка в новом коде, которую необходимо исправить.

ИСПОЛЬЗОВАНИЕ АКТУАЛЬНОЙ ДОКУМЕНТАЦИИ ДЛЯ ПОДДЕРЖКИ СОПРОВОЖДЕНИЯ

- Выполняемые спецификации – это отнюдь не решение всех традиционных проблем технической документации. Нет гарантий, что они всегда будут полезны и значимы — для этого необходимы практика и дисциплина. Для создания полной картины нужны различные виды документации – техническая, архитектурная и функциональная. Однако, если выполняемые спецификации хорошо написаны и организованы, они обеспечиваются значительные преимущества по сравнению с традиционными подходами.

ПРЕИМУЩЕСТВА BDD

- Сокращение потерь
- Сокращение затрат
- Более простое и безопасное внесение изменений
- Сокращение времени на новые релизы

СОКРАЩЕНИЕ ПОТЕРЬ

- BDD – это прежде всего направление усилий разработчиков на выявление и реализацию функций, которые обеспечивают ценность для бизнеса, и исключение тех, которые не обеспечивают. Если команда реализует функцию, которая не соответствует бизнес-целям проекта, то это будет потерей времени и средств для бизнеса. Или, если разработчики реализуют необходимую бизнесу функцию, но так, что она оказывается бесполезной для бизнеса, команде придется переделать эту работу, что также приведет к потерям времени и средств. Методология BDD помогает избежать такого рода напрасных усилий и потерь, позволяя командам сосредоточиться на функциях, которые соответствуют целям бизнеса.
- Кроме того, BDD позволяет сократить потери трудозатрат за счет предоставления пользователям более быстрой и полезной обратной связи. Благодаря этому команда может раньше вносить необходимые изменения.

СОКРАЩЕНИЕ ЗАТРАТ

- Прямым следствием сокращения потерь является сокращение затрат. Направив усилия на разработку функций, имеющих подтвержденную ценность для бизнеса (создание правильного ПО), и не тратя усилия на функции с минимальной ценностью, можно сократить затраты на разработку жизнеспособного продукта для пользователей. А повышая качество кода (создавая правильное ПО), вы сокращаете число ошибок и, следовательно, затраты на их устранение, а также затраты, связанные с задержками по причине этих ошибок.

БОЛЕЕ ПРОСТОЕ И БЕЗОПАСНОЕ ВНЕСЕНИЕ ИЗМЕНЕНИЙ

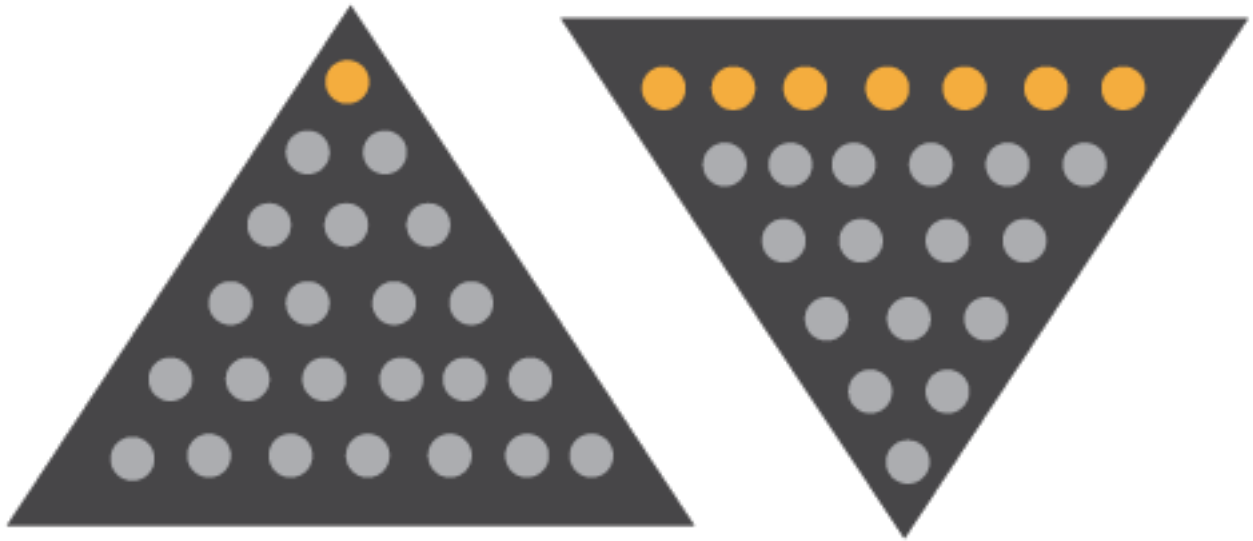
- BDD существенно упрощает процесс изменения и расширения приложений. Актуальная документация создается из выполняемых спецификаций с использованием терминов знакомых стейкхолдерам. Благодаря этому им гораздо легче понять, что именно делает приложение. Кроме того, низкоуровневые выполняемые спецификации выполняют роль технической документации для разработчиков, упрощая тем самым понимание существующей базы кода и внесение изменений.
- И последнее, но не менее важно преимущество: в процессе BDD создается полноценный набор автоматизированных приемочных и модульных тестов, что уменьшает риск регрессий, вызванных новыми изменениями в приложении.

СОКРАЩЕНИЕ ВРЕМЕНИ НА НОВЫЕ РЕЛИЗЫ

- Эти всеобъемлющие автоматизированные тесты также существенно сокращают цикл выпуска ПО. Тестировщикам не нужно бесконечно долго тестировать вручную каждый новый релиз. Теперь они могут изначально использовать автоматизированные приемочные тесты и более продуктивно, с меньшими временными затратами проводить исследовательское тестирование и нестандартное тестирование вручную.

НЕДОСТАТКИ И ПОТЕНЦИАЛЬНЫЕ ПРОБЛЕМЫ BDD

- BDD требует участия и сотрудничества со стороны бизнеса
- BDD лучше всего работает в Agile-командах или итеративном контексте
- BDD не очень хорошо работает в разрозненных командах
- Плохо написанные тесты могут привести к дополнительным затратам на сопровождение



BDD ТРЕБУЕТ УЧАСТИЯ И СОТРУДНИЧЕСТВА СО СТОРОНЫ БИЗНЕСА

- Практики BDD основаны на диалогах и обратной связи. Действительно, диалоги направляют и укрепляют понимание командой требований и способов создания бизнес-ценности на основе этих требований. Если стейкхолдеры не хотят или не могут участвовать в диалогах и взаимодействии, или дают обратную связь только в конце проекта, то будет сложно воспользоваться всеми преимуществами BDD.

BDD ЛУЧШЕ ВСЕГО РАБОТАЕТ В AGILE-КОМАНДАХ ИЛИ ИТЕРАТИВНОМ КОНТЕКСТЕ

- Практика анализ требований в BDD исходит из того, что обычно очень трудно (или даже невозможно) заранее полностью определить требования, и они будут меняться по мере того, как команда (и стейкхолдеры) будут больше узнавать о проекте. Этот подход, естественно, больше соответствует методологии Agile или итеративного проекта.

BDD НЕ ОЧЕНЬ ХОРОШО РАБОТАЕТ В РАЗРОЗНЕННЫХ КОМАНДАХ

- Во многих крупных организациях по-прежнему существует разрозненная структура разработки ПО. Бизнес-аналитики пишут подробные спецификации, которые затем передаются командам разработчиков, обычно работающих вне офиса организации или заказчика. Точно так же, тестирование поручается отдельной команде. В такого рода организациях, тем не менее, возможно использовать методологию BDD на уровне написания кода, и команды разработчиков могут рассчитывать на существенное повышение качества кода, дизайна, обеспечение лучшей сопровождаемости и уменьшение количества ошибок. Однако отсутствие должного взаимодействия между командами бизнес-аналитиков и разработчиками осложняет использование практик BDD для постепенного прояснения и лучшего понимания реальных требований.
- Аналогичные проблемы могут возникнуть и в случае разрозненных команд тестировщиков. Если такие команды ждут завершения проекта или проводят тесты изолированно, то они упускают возможность повлиять на прояснение требований на более раннем этапе, что приводит к лишним трудозатратам на устранение проблем, которые можно было бы выявить раньше и было бы проще исправить. Автоматизация приемочных тестов

ПЛОХО НАПИСАННЫЕ ТЕСТЫ МОГУТ ПРИВЕСТИ К ДОПОЛНИТЕЛЬНЫМ ЗАТРАТАМ НА СОПРОВОЖДЕНИЕ

- Создание автоматизированных приемочных тестов, особенно для сложных веб-приложений, требует определенных навыков, и многие команды, начинающие использовать BDD, сталкиваются с существенными трудностями. Действительно, если тесты недостаточно тщательно спроектированы с нужными уровнями абстракции и выразительными возможностями, то они могут оказаться неэффективными. А если имеется большое число плохо написанных тестов, их сопровождение становится нелегким делом. Во многих организациях успешно применяются автоматизированные тести для веб-приложений, однако для их правильной реализации требуются соответствующие знания и опыт.

ВЫВОДЫ

- Для успешной реализации проекта необходимо создавать надежное и не имеющее ошибок программное обеспечение с функциями, которые имеют реальную ценность для бизнеса.
- В практике BDD используются обсуждения конкретных примеров для достижения общего понимания того, какие функции действительно принесут пользу для организации.
- На основе этих примеров формулируются критерии приемки, которые могут использоваться разработчиками для определения готовности функции.

ВЫВОДЫ

- Приемочные тесты с определенными критериями можно автоматизировать с помощью таких инструментов, как Cucumber, Jbehave или SpecFlow, как для создания автоматизированных регрессивных тестов, так и отчетов, в которых точно описываются функции приложения и их реализация.
- В практике BDD функции реализуются в соответствии с нисходящим подходом, критерии приемки используются как цели, а поведение каждого компонента описывается с помощью модульных тестов, написанных в виде выполняемых спецификаций.
- Основные преимущества BDD: основное внимание уделяется реализации имеющих ценность функций, сокращаются потери времени и затраты, более простое и безопасное внесение изменений, ускорение процесса выпуска ПО.

ГЛАВНЫЕ ВОПРОСЫ ТЕСТИРОВАНИЯ

- С чего начинается процесс тестирования? (контекст - данные и состояние системы)
- Что тестировать, а что нет - фокус тестирования
- Сколько должно быть проверок? (количество тестов)
- Как понять, что тест не прошел? (условие срабатывания)
- Критерии приема (когда завершать тестирование?)

ПРОЦЕСС РАЗРАБОТКИ В CUCUMBER

- Аналитики (или тест-менеджеры) пишут тестовые сценарии на языке, приближенном к естественному.
- Фреймворк преобразует эти сценарии в код с помощью комбинации возможностей Cucumber и Test Runner из Junit.
- Фреймворк связывает ранее сгененированный код сценариев с кодом, который разработчик тестов предполагал для этих сценариев.
- Фреймворк выполняет всю собранную цепочку действий и проверяет результат.

Спасибо за внимание!