**Version Control with Git**

**///**

**Software Development Methodologies**

**SDM CONCEPT**

Software Development Models and Methodologies

(Правила формируются от времени, ресурсов ситуаций используем различные правила)

**Waterfall** – самая старая модель разработки софта.

Особенность WF – идет сверху вниз и не идет в обратном направлении.

1. Анализ требований

Собираются требования. Люди общаются с бизнесом. Проектная команда пытается понять, что делать и как делать. В результате чего генерируется набор требований. (В классическом варианте WF требования не меняются в течении всего проекта)

1. Дизайн

Архитектура. База данных. Прототипирование минимальное.

1. Разработка
2. Тестирование

Планирование

Подготовка тестов

Тестирование и отслеживание проблем

Заключение о качестве

1. Поддержка приложения

Плюсы WF

1. Первая методология разработки софта
2. Простота
3. Каждая фаза начинается после того, как закончилась предыдущая
4. В конце выпускаем продукт

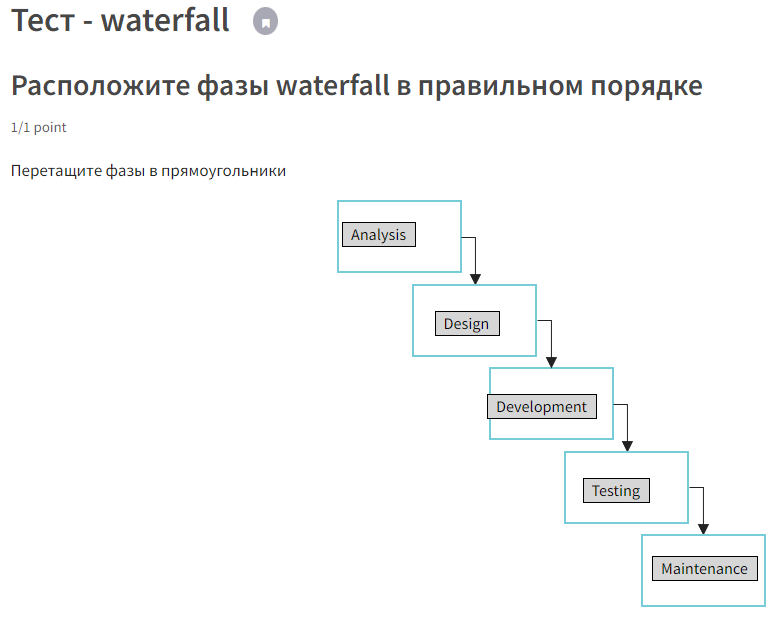
Минусы WF

1. Очень дорого обходятся ошибки (Так как ошибка из-за стадии тестирования может очень долго обнаружатся)
2. Трудно сделать оценку затрат
3. Большое количество документации
4. Сложно оценить время разработки
5. Очень долго ждать конечный продукт

Когда использовать WF

1. Очень высоко регулируемая сфера (Больницы, Военная деятельность, Правительство)
2. Требования хорошо задокументированы, “чисты” и исправлены

ТЕСТ WATERFALL



**AGILE**

Доминирующая в современной разработке.

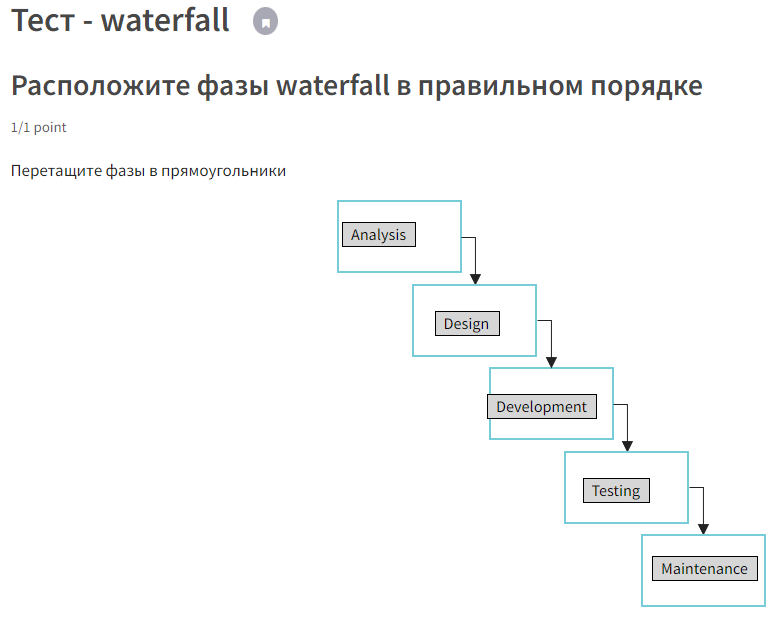
Ценности AGILE

1. Люди и взаимодействие людей важнее, чем процессы и инструменты
2. Рабочий софт, важнее исчерпывающей документации
3. Взаимодействие с клиентом, важнее чем контракт
4. Реакция на изменения важнее, чем следование плану

Принципы AGILE

1. Наивысшим приоритетом является, удовлетворить заказчика с помощью раннего и перманентного сдачи рабочего софта
2. Изменения в требованиях приветствуются
3. Должны доставлять изменения часто (регулярно), не должно быть полгода или год!
4. Люди, которые являются экспертами, и разработчики должны работать вместе ежедневно.
5. Нужно строить вокруг тех, кому это действительно важно. Поддерживать их и дать им то, что нужно.
6. Самый эффективный способ коммуникации – это личное общение.
7. Рабочий софт является мерилом прогресса
8. Процесс должен поддерживать устойчивую разработку
9. Постоянно нужно уделять внимание каким-то техническим практикам
10. Простота важна
11. Самоорганизующуюся команду
12. Команда должна работать эффективно и когда она регулярно показывает какую-либо эффективность, можно вносить вправки в процесс и менять ее поведение.

ТЕСТ AGILE



**SCRUM**

Один из самых популярных фреймворков, который построен на технологии AGILE.

Имеет четкие рамки, роли и элементы.

При разработке приложения у нас есть список требований. Базируется на спринтах (короткий промежуток времени). В течении которого делается часть долей из требований. Берется одна функция и дробится в несколько частей. То, что успевается сделаться за спринт добавляется в SPRINT BRCKLOG. 26 спринтов = 26 версий продукта.

Элементы SCRUM

1. РОЛИ

ВЛАДЕЛЕЦ ПРОДУКТА – отвечает за то, что должно быть в продукте (устанавливает приоритет, планирование, общается с командой)

SCRUM MASTER – человек который отвечает за процесс. Следит за тем, чтобы все участники проекта следовали правилам SCRUM. Решает сложности в процессе возникновения.

TEAM – люди делают продукт. Маленькая команда, которая включает множество разносторонних разработчиков. Нет главы.

1. СОБЫТИЯ

ГРУМИНГ (Backlog refinement) – что будет в спринте. Задает вопросы Владельцу. Решает, как будет сделано и протестировано.

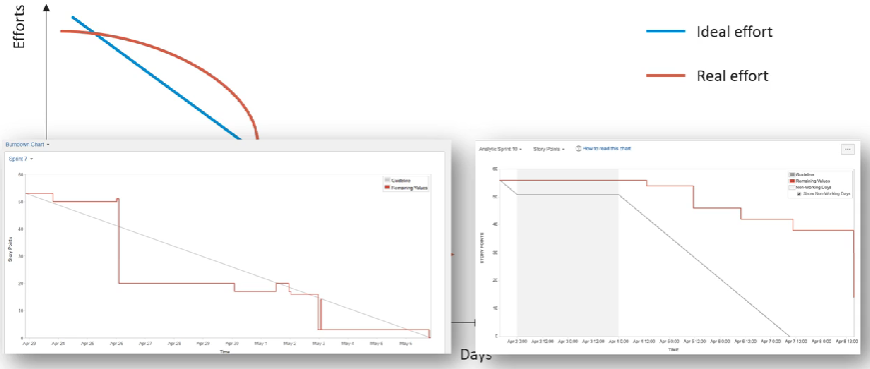
PLANNING – в 1й день спринта. Определяем, что конкретно будем делать. Разбиение больших задач на маленькие. Высокоприоритетные задачи, выбирает задачи для спринта. Даем временную оценку действиям.

STANDUP – Небольшое совещание. Делимся информацией со своей командой. Каждый день все собираются и рассказывают, что сделал сегодня и SCRUM MASTER пытается решить проблему. Лучше личное общение.

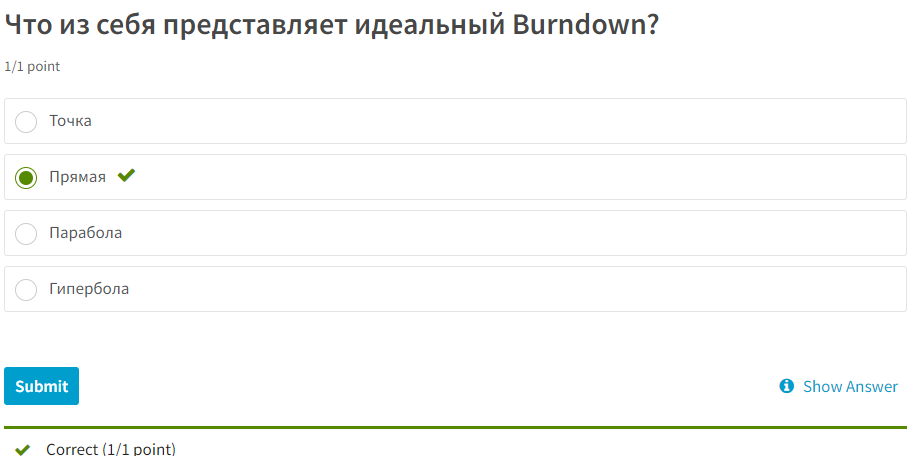
SPRINT REVIEW (DEMO) – показывается, что было сделано. Получаем отзыв на то, что сделано.

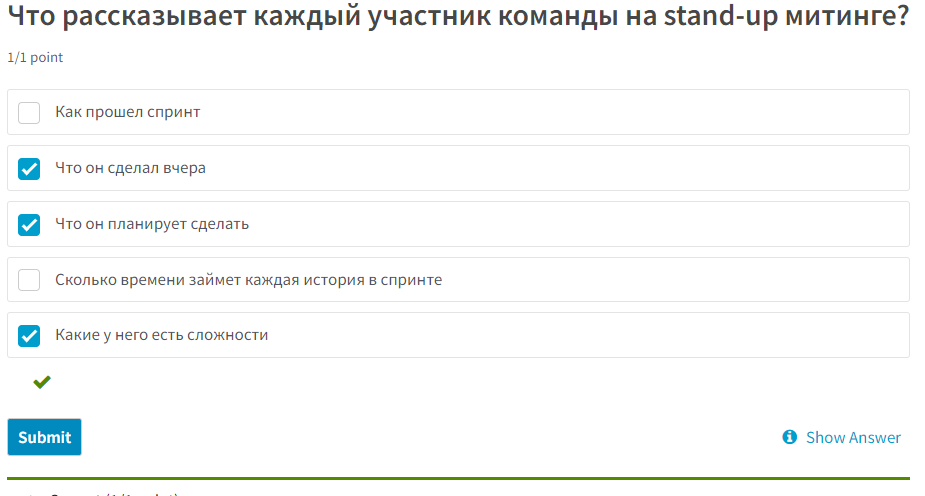
SPRINT RETROSPECTIVE – смотрится, что сделано хорошо. Что можно улучшить. Выделение человека, который отвечает за улучшение. Сроки.

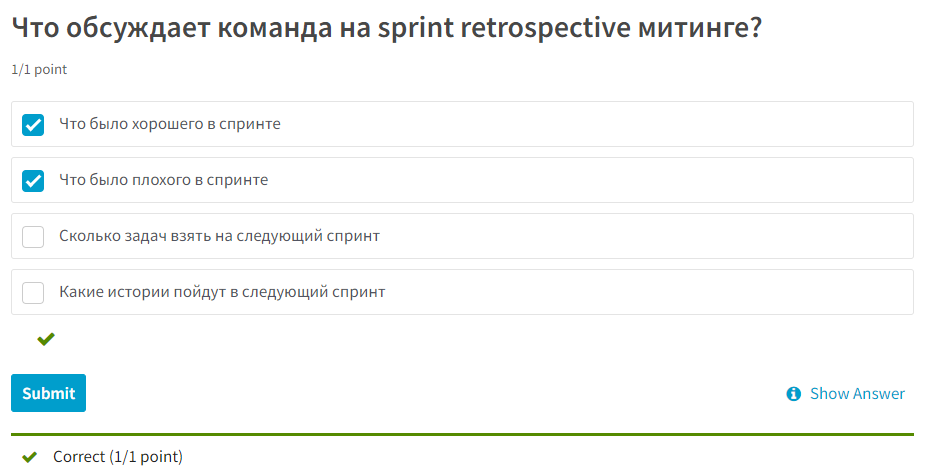
1. АРТИФАКТЫ (PRODUCT BACKLOG (список требований), SPRINT BACKLOG (то, что делается в рамках спринта), PRODUCT INCREMENT (конечный продукт))
2. ДРУГОЕ BurnDown chart. График по одной оси которой время, с другой усилия.



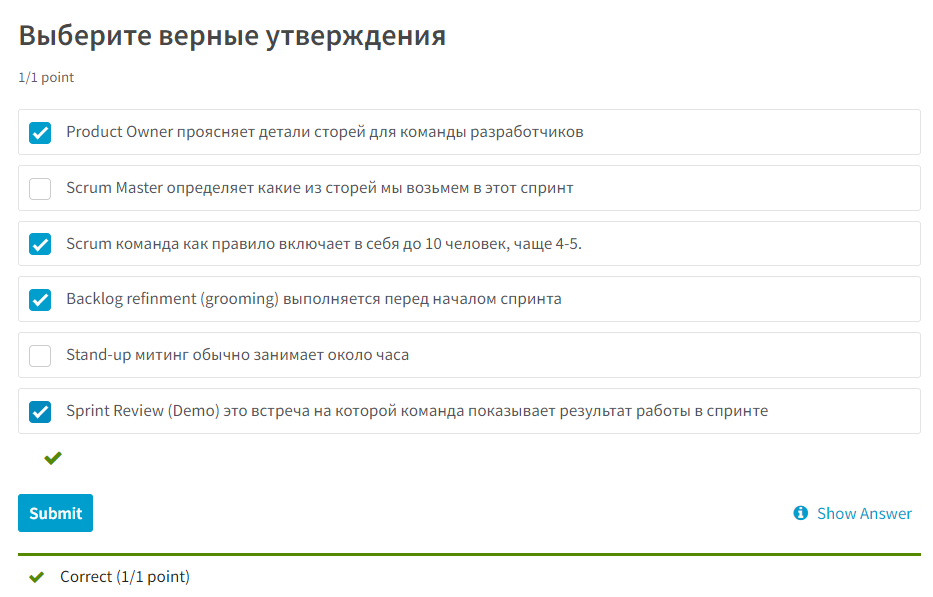
ТЕСТ SCRUM











**KANBAN**

Модель разработки.

ДОСКА.

Доска, на которой находятся задачи. Эти задачи могут перемещаться из одного состояния в другое.

Kanban — это популярный подход к реализации принципов agile и DevOps при разработке ПО.

Задачи визуально представлены на доске Kanban, что позволяет участникам команды видеть состояние каждой задачи в любой момент времени.

Почему канбан?

Не нужно планировать работу на ближайшие недели, вы взяли стикер написали на нем что делать и наклеили на доску. Очень легко начать , можно просто взять пачку стикеров и маркер , за 5 мин объяснить разработчикам что делать.

Подходить для суппорта(поддержки).

**EXTREME PROGRAMMING**

Экстремальное программирование или XP, eXtreme Programming — гибкая методология разработки программного обеспечения.

XP отличается от других гибких методологий тем, что применимо только в области разработки программного обеспечения. Оно не может быть использовано в другом бизнесе или повседневной жизни, как scrum, kanban или lean.

Цель методики XP — справиться с постоянно меняющимися требованиями к программному продукту и повысить качество разработки. Поэтому XP хорошо подходит для сложных и неопределенных проектов

**TDD CONCEPT (TEST DRIVEN CONCEPT)**

Практика экстремального программирования. Сначала делаем тесты потом пишем код.

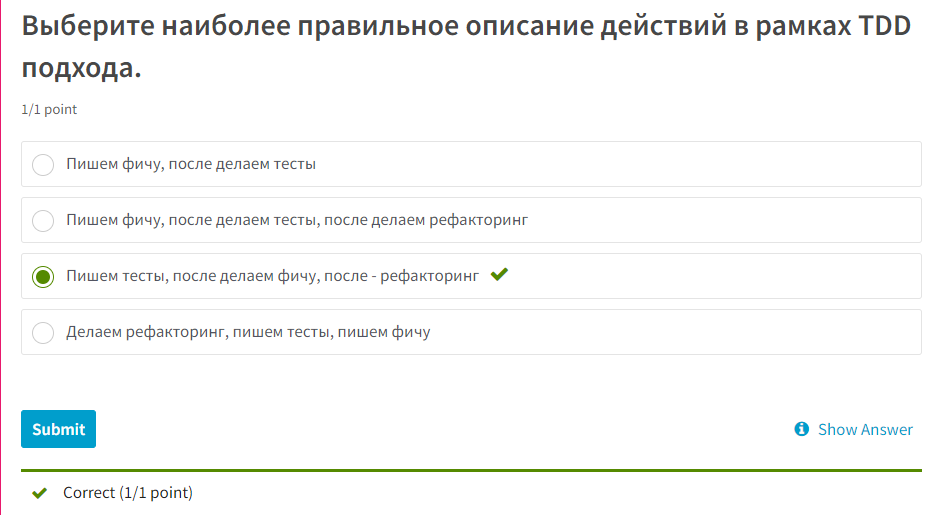
ПОЛЬЗА

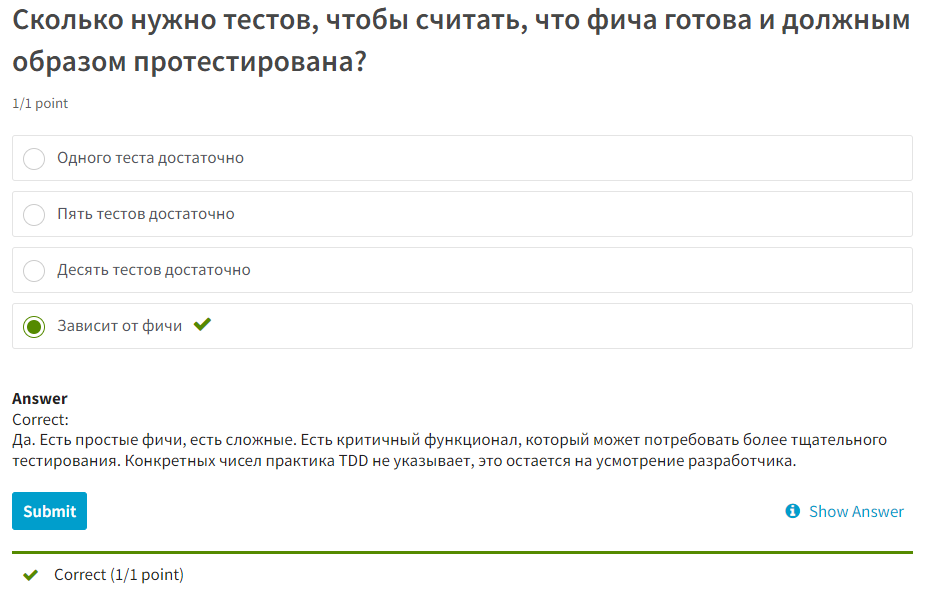
1. Чистый код (задокументированный)
2. Рефакторинг легкий
3. Позволяет оставлять некоторые вещи недоделанными, притом, что программа и без этой недоделки работает нормально

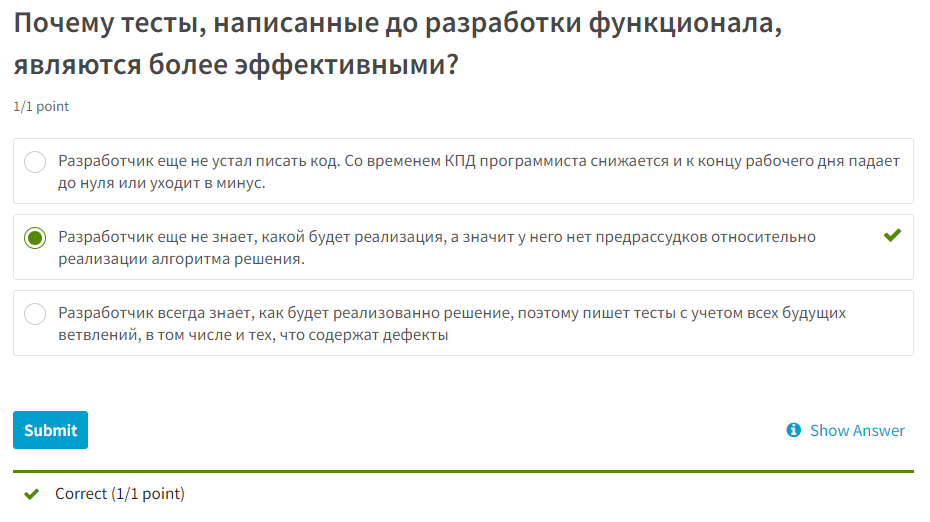
МИНУСЫ

1. Работать тяжелее

ТЕСТ TDD







**BDD**

Бизнес владелец составляет требования. Требования попадают к разработчику и тестостеровщику. Возращается владельцу. И так до конца окончания продукта.

BDD (сокр. от англ. Behavior-driven development, дословно «разработка через поведение») — это методология разработки программного обеспечения, являющаяся ответвлением от методологии разработки через тестирование (TDD).

ЗАКЛЮЧЕНИЕ

AGILE – это принципы, но без четкого определения.

KANBAN, SCRUM – фреймворки которые имеют четкие инструкции

XP, TDD, BDD – практики экстримального программирования связанные со SCRUM и KANBAN.

**INTRODUCTION TO TESTING**

**TEST Strategy part 1**

2 Главных вопроса

Что нам необходимо тестировать?

Какие временные рамки?

ТЕСТОВАЯ СТРАТЕГИЯ отвечает на 3 вопроса

1. Какие тестовые типы у нас будут в рамках тестирования выполнены?
2. Как мы будем выполнять эти типы тестирования?
3. Когда и что будет выполняться в рамках определенных фаз или уровней тестирования?

Типы тестирования (Примеры)

1. Функциональность (Функциональное тестирование – Functional Testing)
2. Правильность данных (Интеграционное тестирование – Integration testing)
3. Как приложение работает на разных устройствах (Compatibility testing)
4. Как наше приложение будет работать во время обычной стандартной функциональности мобильного телефона (например кто-то нам звонит и не закроется или не собьется ли наша операция во время звонка) Работа под различными типами интернет соединения (Mobile Testing)
5. Насколько наше приложение будет защищено от попыток украсть информацию из него (Penetration testing)
6. Насколько быстро будет работать наше приложение (Performance testing)
7. Насколько комфортно будет пользоваться приложением людьми с ограниченными возможностями (Accessibility testing)
8. Поддержка языков (Localization testing)
9. Насколько удобно пользователям использовать это приложение (Usability testing)

**TEST Strategy part 2**

Некоторые типы тестирования нужно(можно) комбинировать:

1. Regression, Localization, Mobile, Compatibility vs Permissions (если есть необходимость проверять доступ пользователей с различными правами)

Некоторые типы тестов требуют человека со специальными навыками или распределением бюджета

1. Performance, security, usability

Некоторые типы тестов обязательны к “специальному тестированию”/бизнес знаний.

1. Accessibility, Localization, Mobile Testing

Какие типы тестирования когда будут выполняться?

1. В конце спринта
2. В середине
3. В конце проекта
4. В начале проекта

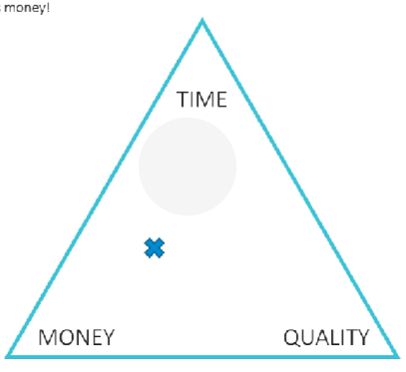
Что надо выяснить для типов тестирования перед началом

1. Среда на которой мы будем выполнять подход, кто ее будет предоставлять.
2. Какие инструменты нам необходимы для тестирования.
3. Кто и когда что выполняет
4. Тестовые данные
5. Тестовая документация
6. Тестовая автоматизация

**AGILE VS SCRUM-FALL**

Как время влияет на подход к тестированию.

Ускорить тестирование можно с помощью автоматизации



2 Течения

БЫСТРЫЕ ВЫПУСКИ ПОСЛЕ КАЖДОГО СПРИТА

1. После каждого спринта мы можем предоставлять новую функцию
2. Все тестирование за автоматизировано
3. Коллаборация разработчиков и тестировщиков
4. Наличие зрелых инженерных практик
5. AGILE

ДОРОЖЕ

ГРУППОВЫЕ СПРИНТЫ В ВЫПУСКЕ

1. КОМБИНАЦИЯ AGILE и WATERFALL
2. Функциональное тестирование выполняется спринтами
3. Дополнительный этап стабилизации перед выпуском
4. Регрессия частично автоматизируется

ДЕШЕВЛЕ НО МЕДЛЕННЕЕ

**TESTING IN AGILE**

Тестирование в AGILE и как его организовать

TRADITIONAL TESTING

Тестировщики обнаруживают разницу между существующими и требуемыми условиями

AGILE TESTING

Тестировщики предотвращают разницу между существующими и обязательными условиями

КАК ПРЕДОТВРАЩАТЬ ДЕФЕКТЫ

1. Начать с тестирования самих требований, до того как приступили к разработке разработчики данного функционала (ГРУМИНГ)
2. Активное общение с разработчиками

2.1) Предоставлять информацию, как можно раньше о том, что мы собираемся тестировать.

2.2) Какого уровня, какого рода тесты, кто будет их создавать (UNIT тесты)

2.3) Код ревью

2.4) Фокусируемся на всем процессе тестирования, разработки и инженерных практик.

3) Разработка автоматических тестов параллельно с разработкой функционала

4) Включение СI / CD чтобы разработчики могли отследить как какое-то изменение повлияло на предыдущий коммит.

Инженерные практики

1. CI/CD
2. Quality Gates (встраивание автоматической проверки кода в свои процессы)
3. Стандарты разработки
4. Код ревью
5. Unit тестирование
6. Приложения для проверки кода
7. Сначала пишем тесты потом код
8. Стратегия релиза (Четкие правила, что должно быть выполнено)

**TESTING in AGILE. AUTOMATION.**

Основополагающим принципом является пирамида.

Как организовать автоматизм в спринте

ДО РАЗРАБОТКИ

Коллаборация с разработчиками для определения какие тесты будут включены, на каком уровне они будут включены и кем. Договоренность с разработчиками о уникальных ID для контролирования (шаблоны запросов/ответов в случае реализации услуг)

ВО ВРЕМЯ РАЗРАБОТКИ

Создание пустых тестов с идеей о тесте.

Организация тестов по шагам: включать возможные шаги и комментарии наших шагов которые зависят от реализации.

Макетные компоненты которые не будут доступны во время спринта.

Генерация(Организация) автоматизированных тестовых данных.

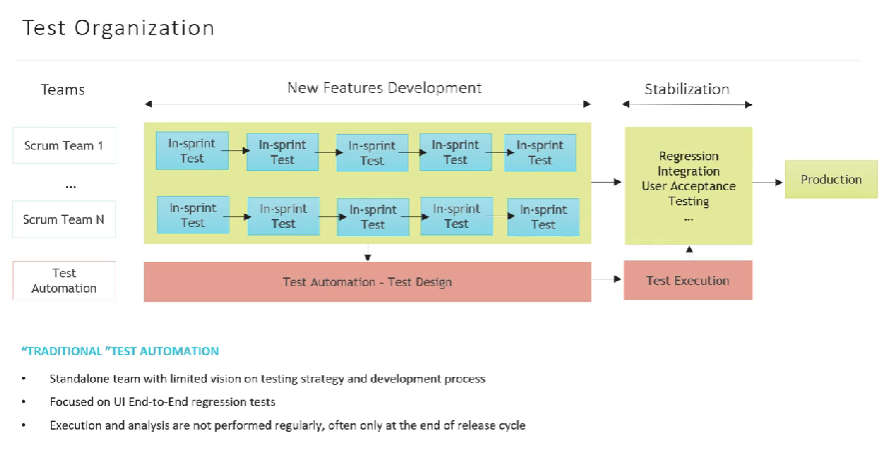
КОГДА РАЗРАБОТКА ЗАКОНЧЕНА.

Отладить новые автоматизированные тесты, сразу же после коммитов.

Запуск всех существующих тестов после последнего коммита.

**SCRUM-FALL**

Как организовывается тестирование в SCRUM-FALL.



ОСНОВНЫЕ ПРОБЛЕМЫ

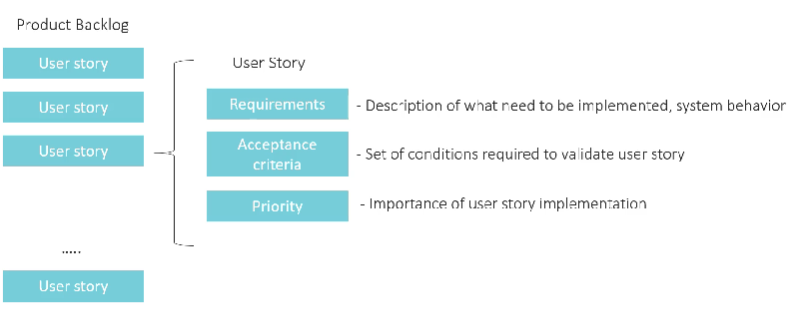
1. Не находит дефекты
2. Не сокращается объем ручного тестирования
3. Тесты не стабильны
4. Выполнение слишком долгое
5. Длительное техническое обслуживание
6. Фидбэк на качество продукта проводится очень долго

КАК РЕШИТЬ

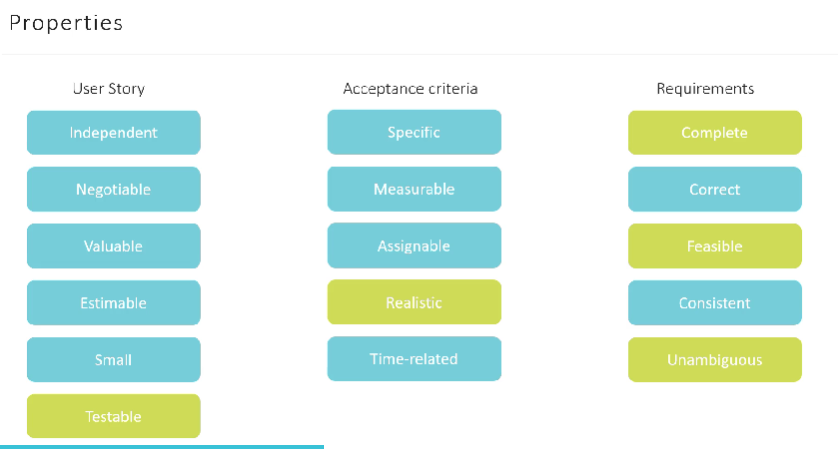
1. Больше взаимодействовать с командой тестирования
2. Комбинация тестирования
3. Включение автоматизированных тестов в CI/CD
4. Параллельное выполнение
5. Выполнение тестов, как минимум раз в спринт

**REQUIRMENT TESTING. INTRODUCTION.**

PRODUCT BACKLOG – набор user сторей.







САМЫЕ ТИПИЧНЫЕ ОШИБКИ В ТРЕБОВАНИЯХ

№1 Неоднозначность

Навигация между скринами должна быстрая

Страница логина должна быть user-friendly

№2 Требования могут противоречить друг другу

№3 Требования могут быть неполными

Начинать работать с требованиями нужно до начала разработки

ГРУМИНГ

Происходит до спринта

Включает команду

**REQUIRMENT ANALYSIS. EXAMPLE.**

Вода

**FUNCTIONAL TESTING.**

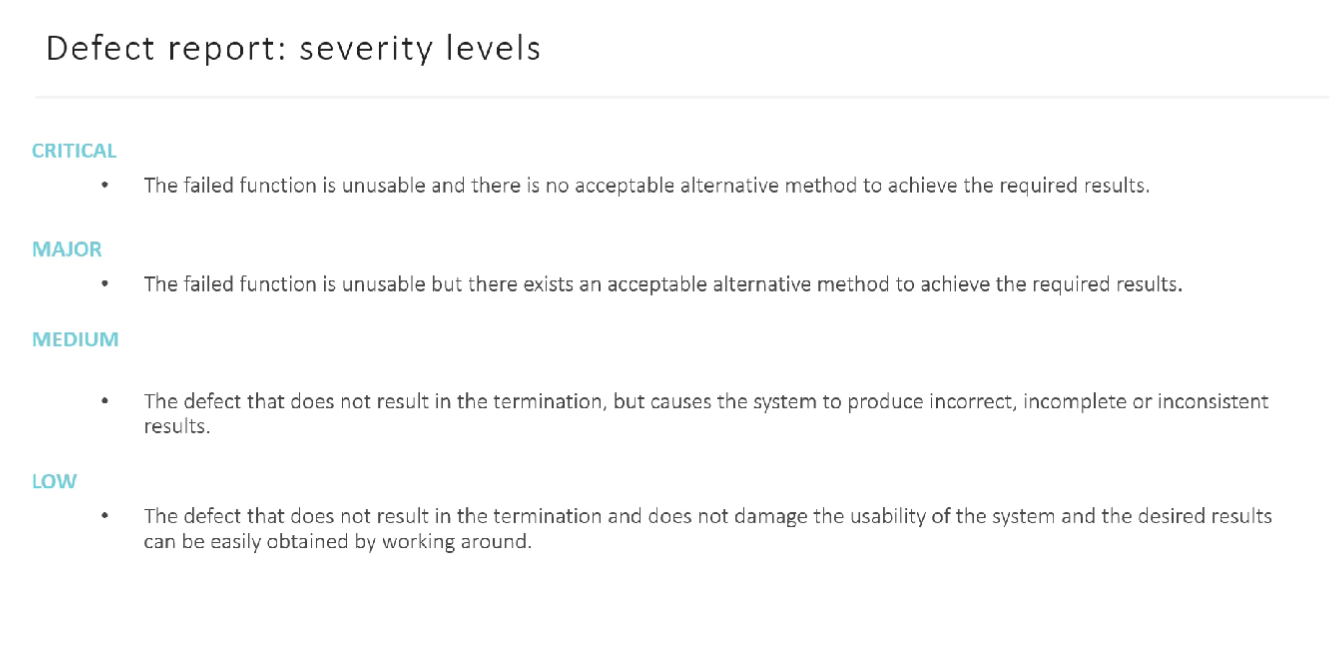
Тестовые сценарии:

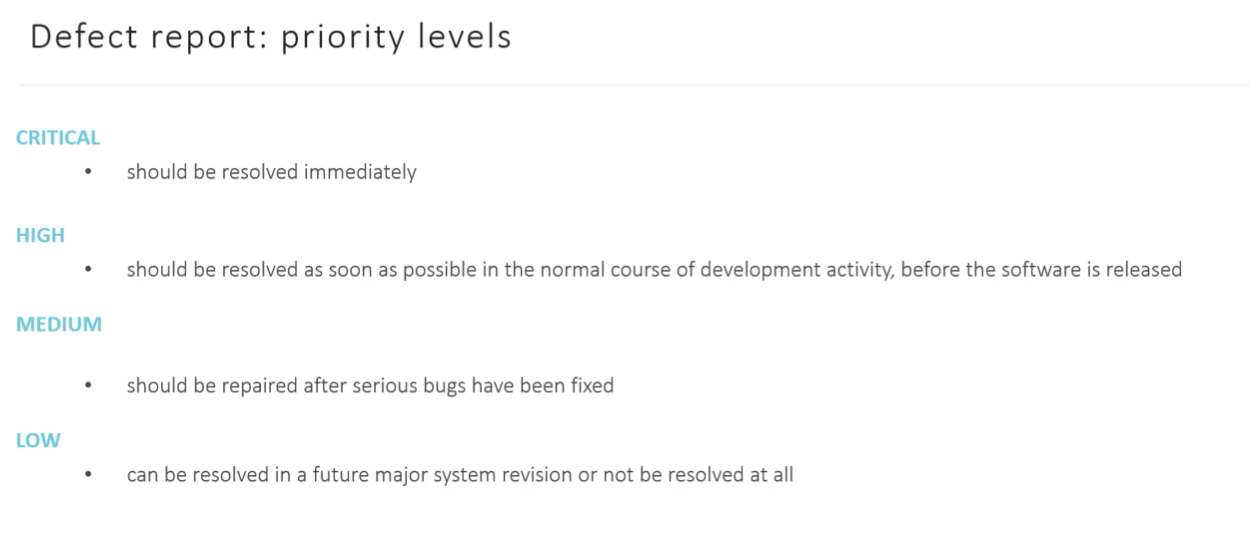
Позитивные

Негативные

**REPORTING DEFECTS.**

Вода





**CONTINIUOUS INTEGRATION WITH JENKINS**