**Waterfall (водопадная модель)**

Есть направление, в котором движется разработка, и она движется только туда(водопад).

1) Анализ требований (собираются требования, люди общаются с бизнесом) ->

2) Дизайн (когда требования собраны, начинаем делать дизайн, т.е. реализовывать архитектуру, базу данных и прототип, формируются общие высокоуровневые решения) ->

3) Разработка (реализация на коде) ->

4) Тестирование (планируем тестирование, подготавливаем системы и прочее для тестирования и делаем вывод исходя из тестирования, сравниваем требования и результат) ->

5) Поддержка (исправление багов, новые патчи, поддержка приложения)

Четко линейная структура. Одна из самым первых методологий.

**Плюсы**: первая модель разработка; простая; все последовательно и логично;  
 **Минусы**: дорого обходятся ошибки; тяжело делать оценку (делается на стадии сбора требований); много документации;

**Когда использовать**: требования хорошо описаны и задокументированы, чисты и пофикшены, строго регулируемая область (правительство, медицина, образование).

**Agile**

Является доминирующей в современной разработке. Говорит будьте гибкими, общайтесь друг с другом.

**Принципы**: удовлетворить заказчика быстрой разработкой софта; изменения в требованиях приветствуются (должны быстро реагировать на изменения); должны быстро работать; работать надо вместе (общаться с разработчиками, с заказчиками и следить за разработкой); нужно строить проект вокруг тех, кому это важно; самый эффективный способ коммуникации – лицом к лицу; рабочий софт – мерило прогресса (оценить работы команды можно по тому, как они сделали софт); agile процесс должен поддерживать устойчивую разработку (должна быть четкая скорость работы, с высокой, но не перенапрягаться); постоянно уделять внимание хорошим техническим практикам и хорошему дизайну, чтобы не накапливался технический долг; простота – важна (делай проще, команда должна работать эффективно;

**Плюсы**: люди и взаимодействие людей важнее, чем процессы и инструменты; рабочее ПО важнее исчерпывающей документации; взаимодействие с клиентом важнее, чем контракт, быстрая реакция на изменения (в законе, выход новой технологии);

**Минусы**: недостаточная документация, неясные требования (подразумевает постепенное выяснение требований и постоянное взаимодействие с заказчиком), риски в отношениях качества (подразумевает быстрые циклы разработки и частую поставку рабочего кода)

**Когда использовать**: проекты с невысокой степенью определенности требований;

проекты с высокой степенью инноваций; команды с высокой самоорганизацией; ситуации, требующие гибкости и масштабируемости;

**Scrum**

Имеет четкие мероприятия, роли и элементы. Работа в scrum фиксируется на спринтах (какой-то короткий промежуток времени – неделя или 2 недели), в течение которой мы будем делать определенные части из backlog’а.

1) Когда начинаем разрабатывать приложение, есть список требований (того, что мы хотим видеть в ПО) – product backlog ->

2) определяем, что мы успеем сделать, что не успеем за спринт, и берем для разработки, либо же разобьем задачу на подзадачи, и что успеем сделать в промежутке спринта, добавляем в sprint backlog ->

3) проходит итерация (спринт) ->

4) есть некий продукт, который готов к релизу

Scrum можно разделить на несколько составляющих:

1) Роли: (

**product owner** (обычно 1 человек) – поставляет story (что делать - требования) разработчикам, является ответственным человеком, который отвечает за первоначальное планирование, говорит, что брать на спринт и управляет backlog’ом, устанавливает приоритеты в планировании;

**scrum master** – человек, который отвечает за процесс scrum (как правило, один из разработчиков), способствует, чтобы все участники команды действовали по принципам scrum’а, решает сложности, которые возникают во время scrum;

**development team** – команда, которая разрабатывает продукт, в scrum это маленькая команда (3 – 9 человек), команда организована самостоятельно и работает, как единая команда, внутри есть все нужные люди (дизайнеры, разработчики, тестировщики, бизнес - планеры));



2) мероприятия: (

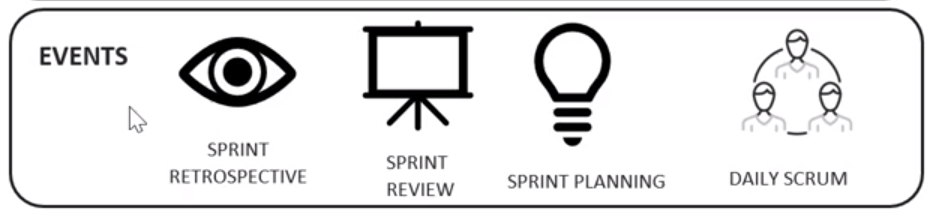
**Sprint grooming (backlog refinement)** – мероприятие, когда мы выбираем, что будем в спринте, происходит перед спринтом, нужно задавать вопросы и много говорить – надо четко понять, что требуется в текущем спринте;

**Sprint planning** – планирование (1-ый день спринта), определяем, что конкретно мы будем делать в течение спринта, разбиение больших задач на маленькие, оцениваем, сколько времени уйдет на подзадачу, начинаем с высоко приоритетных требований и разбиваем их на задачи и делаем предварительную оценку, за сколько выполнишь текущую задачу;

**Daily scrum** – небольшие совещание (5-тиминутка), каждый день scrum-команда собирается вместе и каждый говорит, что сделал вчера и что планирует сегодня, scrum master слушает все проблемы, записывает и постарается решить данное затруднение;

**Sprint review / demo** – разработчик показывает, что он сделал, получает ответ от разработчика (хорошо сделал или нет), происходит раз в sprint;

**Sprint retrospective** – проработали спринт и смотрим на то, что понравилось разработчикам в данном спринте, что можно улучшить, выделяем ответственного человека, который будет отвечать за это улучшение, установка сроков улучшения);

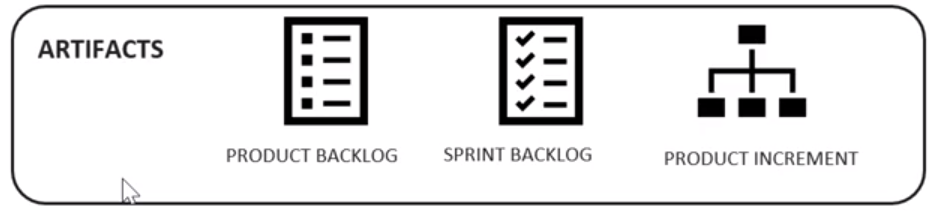


3) артефакты: (

**product backlog** – список всех требований;

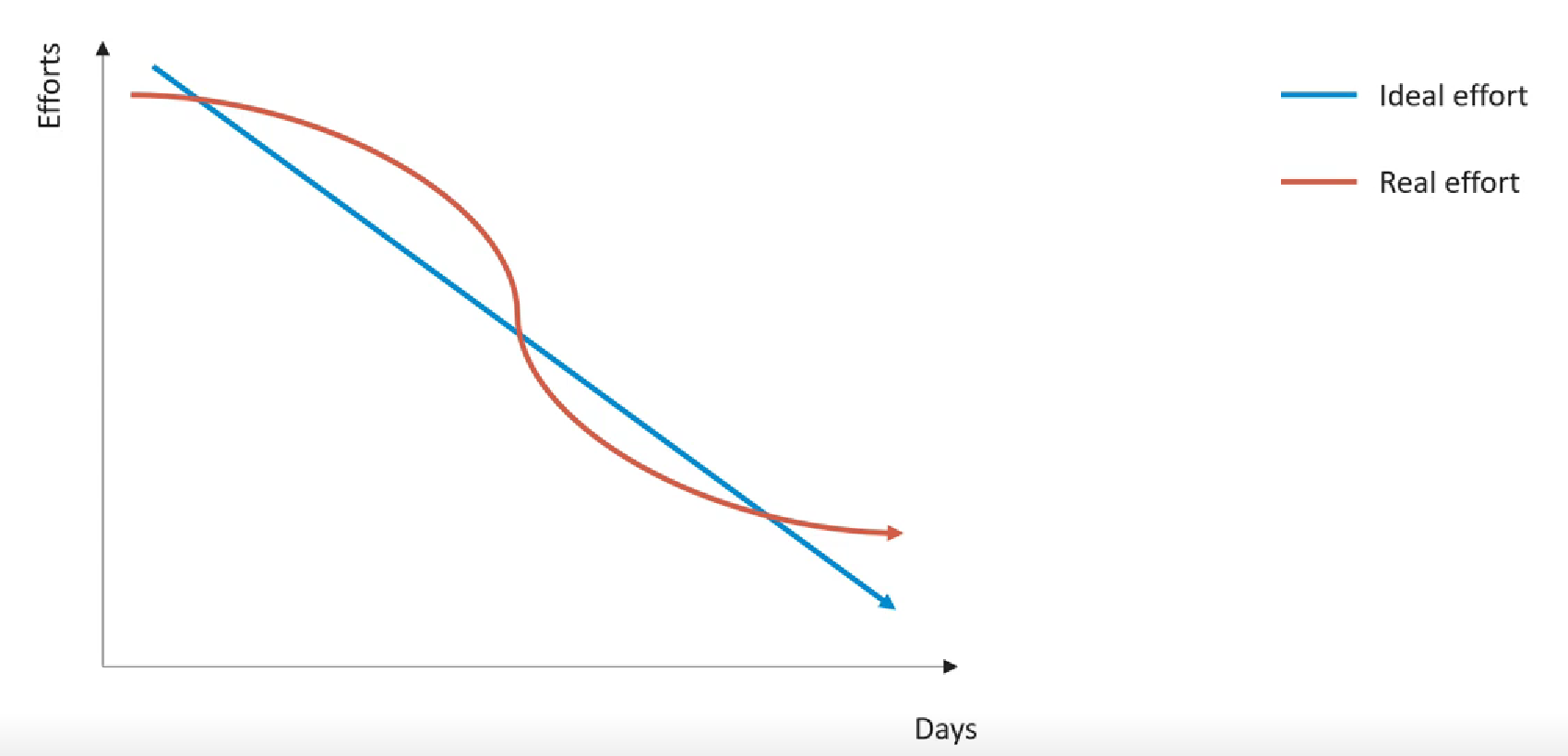
**sprint backlog** – то, что будем делать в рамке спринта,

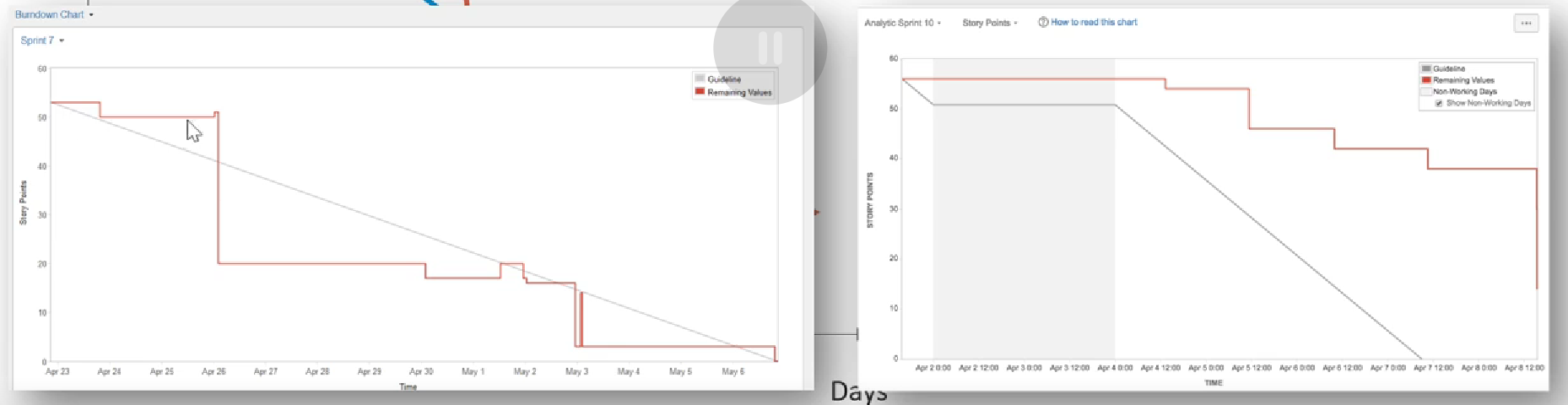
**product increment** – готовая часть продукта);

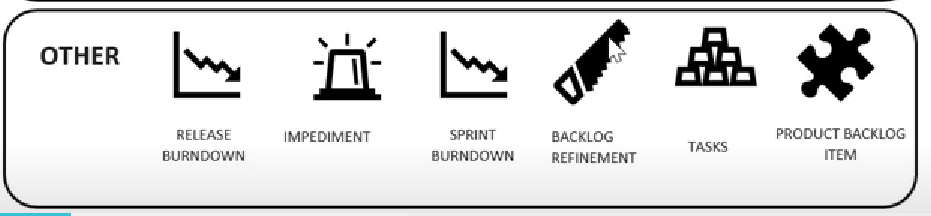


4) прочее: (

**Burndown chart** – график – время на efforts (усилия, ресурсы),

 -> сам график,

 -> реальные проекты,)



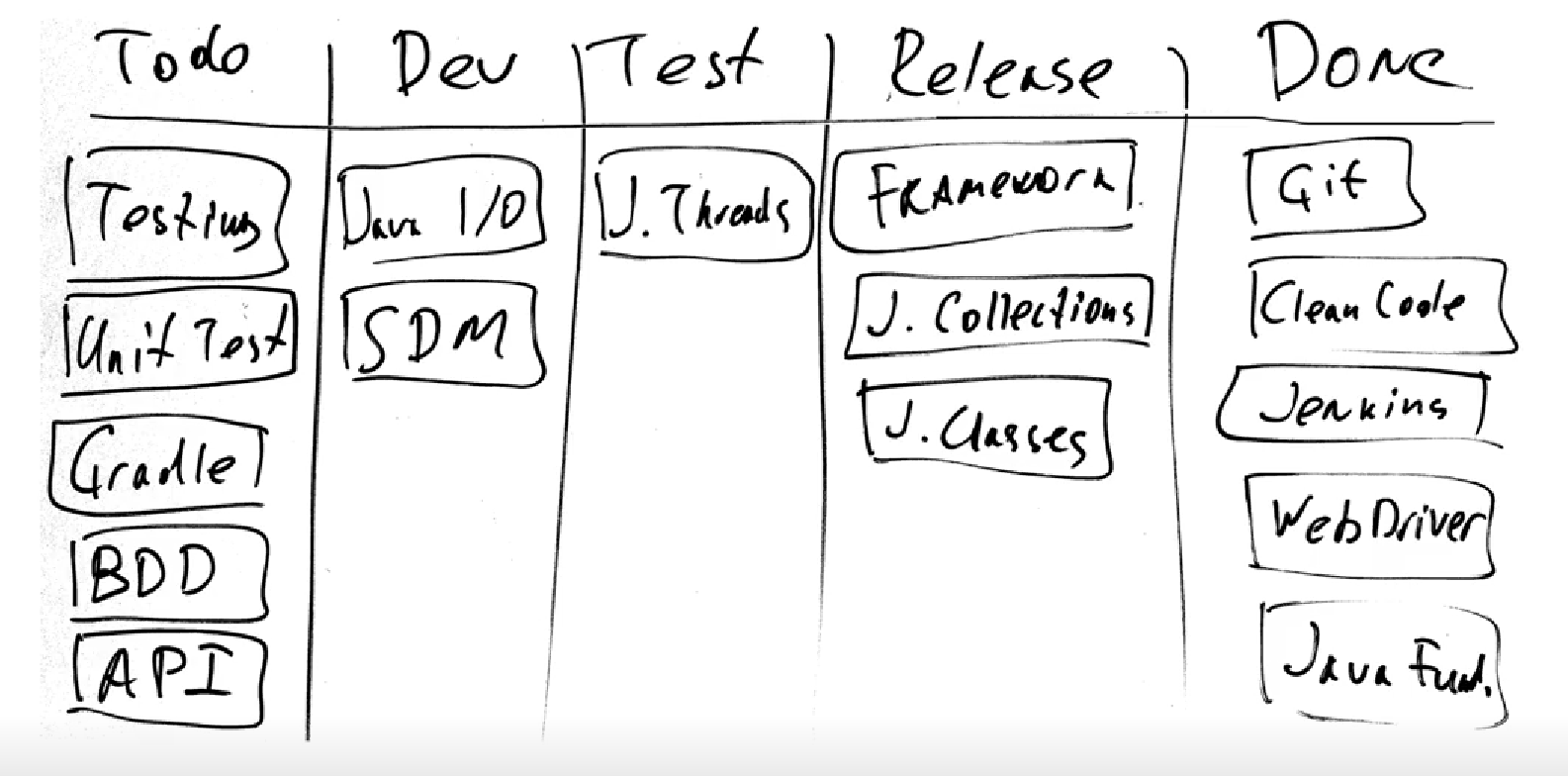
**Плюсы**: гибкость, прозрачность действий во время разработки, быстрая доставка и ценности;

**Минусы**: зависимость от командной работы, не подходит для всех проектов, необходимость постоянного взаимодействия с заказчиком;

**Когда использовать**: когда требуется гибкость и быстрая адаптация к изменениям требований, когда проект имеет высокий уровень неопределенности и требует итеративного подхода, когда важно обеспечить прозрачность и частую обратную связь с заказчиком.

**Kanban**

Основной элемент: доска (Kanban board), на которой находятся задачи. Нет жестких рамок. Переносить задачи постепенно с одной колонки в другую.



**Плюсы**: не нужно планировать действия на неделю вперед, есть хорошая визуализация, легко начать работать;

**Минусы**: отсутствие жесткой структуры, отсутствие строгой приоритизации, зависимость от самоорганизации команды;

**Когда использовать**: для поддержки приложения (моментально ответить на запрос), есть зависимость от другой команды, если есть много изменений.

**Extreme programming**

Основная задача: улучшение процесса разработки продукта. Основные принципы ЭП:

1) Planning Game (собираются несколько человек и планируют)

2) Short Release (делаем часто маленькие кусочки)

3) Metaphor (сложные вещи нужно объяснять простыми словами, примерами)

4) Simple Design (изначально надо сделать простой концепт приложения, а потом его усложнять)

5) Testing: unit tests, automated tests (тестирование)

6) Refactoring (практика по улучшению реорганизации кода без изменения его функционала)

7) Pair Programming (парное программирование, некоторые компоненты вместе кодировать)

8) Collective Ownership (коллективная собственность - весь проект, любой разработчик может менять любую часть проекта)

9) Continuous Integration (практика непрерывной интеграции)

10) 40 hours Week (40-часовая рабочая неделя)

11) On-site Customer (эффективно, когда клиент находится рядом с процессом разработки)

12) Coding Standards (должны поддерживать стандарты кодирования)

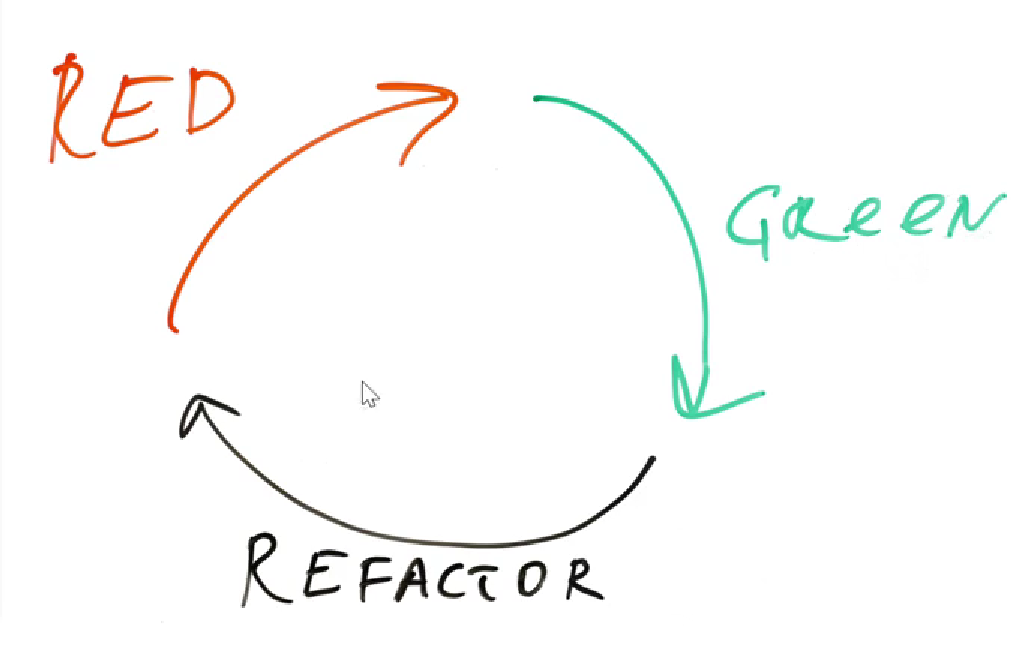
**Плюсы**: высокое качество кода, гибкость и адаптивность, коллективная ответственность, быстрая обратная связь;

**Минусы**: зависимость от командной работы, необходимость подержи заказчика

**Когда использовать**: когда проект требует высокого качества кода и акцентирует внимание на тестировании и регулярном рефакторинге, когда требуется гибкость в адаптации к изменяющимся требованиям и регулярной коммуникации с заказчиком, когда в команде есть хорошая командная динамика и способность сотрудничать над проектом.

**Test-Driven Development**

Одна из практик экстремального программирования. Сначала делаем тесты, а потом пишем код (описываем то, как должен работать алгоритм).



1) Создаем тесты

2) Они все fail (Red)

3) Сделать их зелеными (Green)

4) Переписывание кода (чтоб был масштабируемый, переносимый и красивый) (Refactor)

**Плюсы**: красивый код, легко рефакторить код, меньше дефектов

**Минусы**: тяжело работать, занимает больше времени и усилий, зависимость от качества тестов

**Когда использовать**: когда требования к проекту хорошо определены и понятны, если проект содержит сложную логику или алгоритмы, быстрая обратная связь.

**Behaviour-Driven Development**

1) Business owner общается с аналитиками и говорит, что он хочет

2) Business owner и аналитик пишут сценарий на языке Gherkin

3) Эти сценарии используются всеми: разработчиками, тестировщиками, автоматическими тестами

4) Готовый продукт идет к Business owner’у.

Отличие от TTD в том, что Business owner сам пишет тесты, используя документацию на языке Gherkin.

**Плюсы**: улучшенное понимание требований, улучшенная автоматизация тестирования, улучшенная коммуникация и совместная работа;

**Минусы**: дополнительные затраты времени и ресурсов, сложность внедрения сложность поддержки и обновления сценариев поведения, потребность в постоянных встречах с заказчиком;

**Когда использовать**: когда важна коммуникация и совместная работа, когда необходимо автоматизировать тестирование, когда важен пользовательский опыт

Документ, написанный на Gherkin, является и документацией, и тестами и сценарий для автотестирования.

**Заключение и основные выводы**

Agile - методология, которая имеет ценности и принципы, но не дает четких инструкций, что делать.

Kanban и Scrum – методологии, которые имеют четкие инструкции, что делать.

Есть практики Extreme Programming, TDD, BDD которые можно легко комбинировать с другими методологиями.