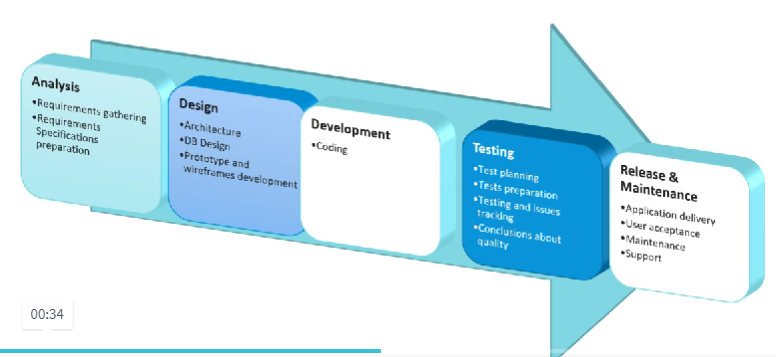
**Модели разработки софта**

**Waterfall (водопадная модель)** – идет сверху вниз, и обратно не идет.



1. Анализ (команда пытается понять, что и как делать)
2. Когда требования собраны, идет стадия дизайна
3. Фаза девелопмента, пишется код (самая продолжительная)
4. Тестирование и делаем вывод – какого качество продукта?
5. Поддержка (отправили заказчику проект, может какие-то изменения)

**Особенности:**

1. Очень простой, один и первых методов.

**Недостатки:**

1. Очень дорого обходятся ошибки (3 пункт может идти очень долго, и только спустя время тестирование выявляет какие-то ошибки). Дорогие в плане количества времени.
2. Может быть переоценка времени работы.

**Agile (эджайл)** – говорит как делать в случае затруднений.



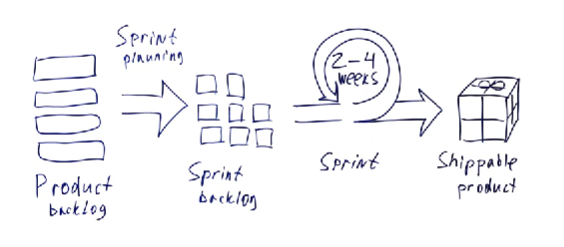
**Ценности:**

1. Люди и взаимодействие людей важнее чем процессы и инструменты;
2. Рабочий софт важнее исчерпывающей документации.
3. Взаимодействие с клиентом (типо дружеские) важнее чем контракт.
4. Реакция на изменения (!!!)

**Набор принципов (продолжение ценностей):**

1. *Высший приоритет* – удовлетворение заказчика, с помощью раннего деливери софта.
2. Изменения в требованиях приветствуются.
3. Изменения должны быть часто (~1-4 недели).
4. Разработчики и бизнес должны работать вместе.
5. Нужно строить проект вокруг тех, кому это действительно важно.
6. *Личное общение* – самый эффективный способ коммуникации.
7. Рабочий софт – показатель прогресса. Главная метрика.
8. Процесс должен поддерживать устойчивую разработку. Четкая скорость работы.
9. Постоянно уделять внимание хорошим техническим практикам и дизайну.
10. Простота.
11. Самоорганизующаяся команда.
12. Команда должна работать эффективно и только после этого вносить правки в процесс.

**Scrum concept (Скрам)**



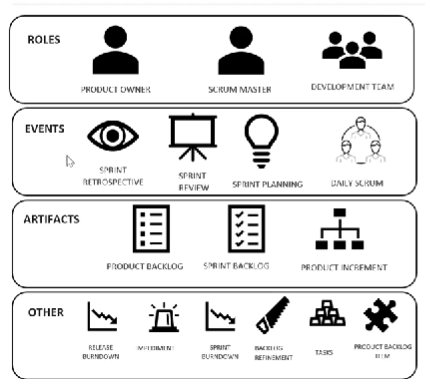
**Как работает:**

1. *Product backlog* . У нас есть список фитчей (Что хотим видеть в продукте) ;
2. *Sprint planning*;
3. *Sprint backlog*. Что мы успеем сделать в рамках спринта (какие фитчи);
4. *Sprint* Дальше операция (может быть неделя, месяц и т.д.);
5. *Shippable product*. Конечный продукт, который мы готовы деливерить;

**Преимущества:**

1. Спринты идут один за одним.
2. Четко прописано как это делать и кто что делает.

**Элементы скрама:**



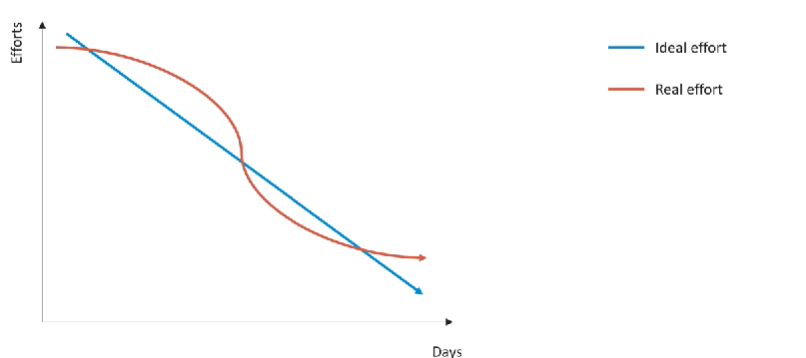
1. **Roles (Роли):**

* **Product Owner (Владелец)** – решает что будет в продукте (поставляет стори) и приоритет выполнения. Управляет *backlog*-ом. Общается с основной командой.
* **Scrum master (один из разработчиков)** – отвечает за скрам, процесс. Следит, чтобы все участники правильно делали работу. Помогает с точки зрения процесса. Решает какие-то сложности.
* **Development Team (команда)** – те, кто выполняет полный цикл разработки и тестирования. Маленькая команда (3-9 человек +-).

1. **Events (Мероприятия):**

* **Sprint Grooming (backlog refinement)** – мероприятие, когда выбираем, что будет у нас в следующем спринте. Обычно последний день спринта. Команда выбирает требования к задачам. Собираются вопросы *Product Owner-у.* Задаем вопросы и много говорим. Необходимо четко понять, что от нас хотят.
* **Sprint Planning** (планирования) – обычно 1ый день спринта. Определяем, что конкретно будет делать в сторях (требованиях). Возможно разбиение задач на более мелкие таски. + Оценка количества времени выполнения тасков. Начинаем с приоритетных сторей.
* **Daily scrum / standup (пятиминутка, 10-15 минут)** – каждый день команда собирается и рассказывает, кто что сделал вчера, планирует сегодня и какие есть проблемы.
* **Sprint Review / Demo** – показывают, что сделали. Может и не быть (если у клиента нет возможности посмотреть). Происходит раз в 1 спринт.
* **Sprint Retrospective** – анализ спринтов, что понравилось, что нет. Что можно улучшить? Сроки.

1. **Other (Остальное): Burndown chart –** график**, Oy** усилия **(effors), Ox** время **(days).**



Чем больше дней, чем меньше осталось сделать.

**Kanban** – проще, чем скрам. Основной элемент доска (борда).

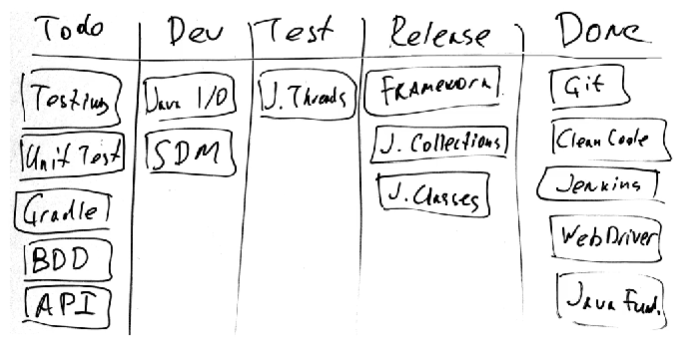
To do - сделать

Developer – разработка

Test - Фаза тестирование

Release - готовы к релизу

Done - сделаны и выгружены



**Почему канбан?**

1. Не нужно продумывать скоуп работы. Взяли, написали что нужно сделать и наклеили на борду.
2. Классная визуализация. Все понятно.
3. Очень легко начать.

**Для кого подходит?**

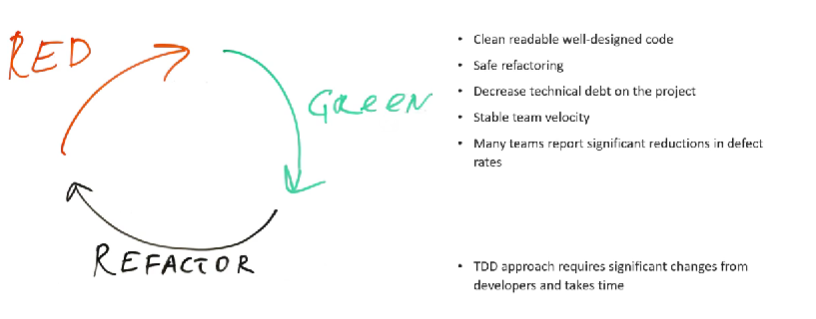
1. Для саппорта (поддержки).
2. У нас есть много изменения в *backlog*-е во время спринта. И мы можем начать делать по канбану.
3. Или есть приходящая/уходящая команда, они будут работать по канбану.

**XP (Extreme Programming) –** улучшение процесса разработки продукта**.**

**Основные моменты:**

1. **The Planning Game** – как и в скраме, собираются несколько человек и делают планирование.
2. **Short Release** – короткие релизы, делаем часто маленькие кусочки.
3. **Metaphor** – метафора, сложные вещи необходимо объяснять простыми словами.
4. **Simple Design** – простой дизайн.
5. **Testing** – юнит тесты и автоматизированные тесты.
6. **Refactoring** – практика по улучшению организации кода, без изменения его функционала (изм структуры, чище, масштабируемым).
7. **Pair Programming** – парное программирование. Коллективное принятие кода.
8. **Collective Ownership** – коллективная собственность. Любой девелопер может менять часть кода.
9. **Continuous Integration** – непрерывная интеграция.
10. **40 hours Week** – программисты могут выходить (овертаймить) на внеочередную работу, но ненадолго.
11. **On-site Customer** – когда клиент находится рядом с программистами. Кастомер – тот, кто продукт оунер.
12. **Coding Standards** – мы должны использовать стандарты кодирования.

**TDD (Test Driven Development)**



Сначала делаем тесты, а потом уже пишем код.

**Как работает:**

1. Создаем тесты, которые будут тестировать этот код. (ОПИСЫВАЕМ КАК ДОЛЖНА РАБОТАТЬ СИСТЕМА)
2. Тесты падают. Становятся красными.
3. Делаем их зелеными.
4. Делаем код красивее.

**Какая польза?**

1. Чистый, задокументированный и читаемый код.
2. Гораздо легче вносить изменения с тестами.
3. Подтверждение эффективности работы в нахождении деффектов.

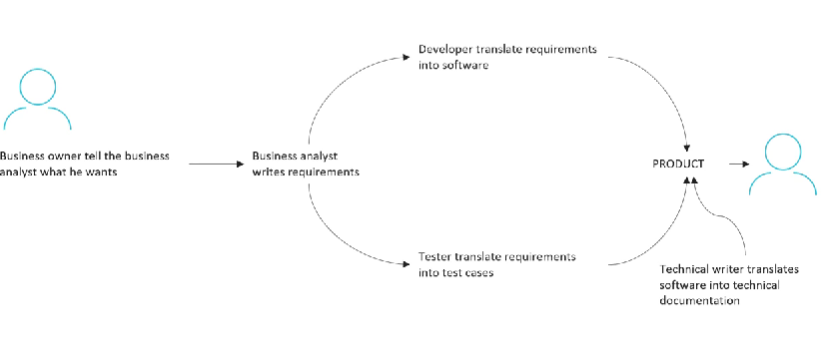
**Минусы?**

1. Это сложно…. Много времени и ресурсов.

**BDD (Biheivor Driven Development)**

**Non-BDD**

1. **Business owner** говорит **бизнес-аналитику** что он хочет.
2. **Бизнес аналитик** пишет требования.
3. Требования попадают к **девелоперу** и **тестировщику**. Разработчик сделал продукт, тестировщик проверяет удовлетворяет ли требованиям.
4. Конечный **продукт**
5. Он возвращается к **Business owner**.



**BDD**

1. **Business owner** общается с аналитиком и говорит что хочет.
2. Далее они вдвоем или кто-то один пишет **сценарии** на языке джеркин.
3. Джеркин спецификации являются условием принятия, сделал или не сделал. Базой для тестов тестировщика и для автоматизации. То есть все эти сценарии используются всеми.
4. Юнит тесты составляются аналитиком.

