**Каскадная методология (Waterfall)**

**I. Основы:**

1. **Каскадная модель (Waterfall):** Представьте строительство дома. Сначала создают *полный* проект (фундамент, стены, коммуникации), затем *строго последовательно* заливают фундамент, возводят стены, проводят коммуникации, делают отделку. Перейти к следующему этапу можно *только после полного завершения предыдущего*. Так работает Waterfall: проект разбит на четкие, линейные этапы. Возврат назад сложен и дорог.
   * **Основные этапы:**
     1. Сбор и анализ требований (что сделать?)
     2. Проектирование (как сделать?)
     3. Реализация (разработка/строительство)
     4. Тестирование (соответствует ли требованиям?)
     5. Внедрение и сопровождение.
2. **Waterfall vs Agile:**
   * **Преимущества Waterfall:** Четкое планирование (сроки, бюджет), понятная документация, простота управления, предсказуемость (при стабильных требованиях), подходит для регулируемых отраслей.
   * **Недостатки Waterfall:** Низкая гибкость (изменения очень дороги), позднее обнаружение ошибок (особенно в требованиях), долгое ожидание работающего продукта, риск неактуальности результата.
3. **История:** Формально описана Уинстоном Ройсом (Winston W. Royce) в 1970 г. в статье "Managing the Development of Large Software Systems". Хотя Ройс критиковал чистый Waterfall и предлагал итерации, именно последовательная модель стала стандартом. Истоки - в традиционном инженерном проектировании.

**II. Практическое применение:**

1. **План проекта (структура):**
   * Введение (цели, участники)
   * Детальное описание *каждого этапа* (задачи, сроки, ресурсы, ответственные, выходные документы)
   * График (Диаграмма Ганта)
   * Бюджет
   * Управление рисками
   * Процедура управления изменениями
   * Приложения (шаблоны документов).
2. **Обязательные документы (по этапам):**
   * *Требования:* Техническое задание (ТЗ), Детальные спецификации.
   * *Проектирование:* Технический проект (ТП), Архитектурное описание, Спецификации интерфейсов.
   * *Реализация:* Код, Техдокументация.
   * *Тестирование:* План тестирования, Тест-кейсы, Отчеты об ошибках, Отчет о тестировании.
   * *Внедрение:* План внедрения, Руководство пользователя, Акт приемки.
3. **Инструменты:**
   * **MS Project:** Лучший для детальных графиков (Ганта), управления ресурсами/бюджетом.
   * **Excel:** Для расчетов, списков, простых графиков.
   * **Jira (в классическом режиме):** Подходит для трекинга задач по этапам, но менее удобен для сложных зависимостей и графиков, чем MS Project. Часто требует жесткой настройки.

**III. В разных сферах:**

1. **Где применяется:** Строительство, Аэрокосмическая отрасль (NASA, Boeing), Оборона, Крупные госпроекты, Производство сложного оборудования, Фармацевтика (регуляторика). Там где требования стабильны, риски высоки, нужна полная документация.
2. **Waterfall в IT:** Не всегда "устаревший". **Выбор зависит:**
   * *Подходит:* Для проектов с четкими, неизменными требованиями (миграции, интеграции), строгой регуляторикой (финансы, здравоохранение), очень маленьких/простых проектов.
   * *Не подходит:* Для проектов с неясными/меняющимися требованиями, инновационных продуктов, где нужна быстрая обратная связь от пользователей.
3. **Гибриды (Water-Scrum-Fall):** Комбинируют этапы. Пример:
   * *Waterfall на верхнем уровне:* Планирование, утверждение бюджета, релиз - линейно.
   * *Agile (Scrum) на уровне разработки:* Реализация и тестирование функциональности ведется итерациями (спринтами) внутри этапа "Реализация". Требования для спринтов берутся из ТЗ этапа "Проектирование".

**IV. Проблемы и решения:**

1. **Ошибки в требованиях на тестировании:**
   * *Решение:* Формальный запрос на изменение (RFC), оценка влияния на сроки/бюджет, утверждение заказчиком, возврат на этап требований/проектирования -> дорого и долго.
   * *Минимизация рисков:* Максимально детальный сбор требований на старте, активное вовлечение заказчика, прототипирование, формальные процедуры проверки (ревью) ТЗ.
2. **Управление изменениями:** Через строгий **Процесс управления изменениями (Change Control Board - CCB):**
   1. Подача запроса на изменение (RFC).
   2. Анализ влияния (сроки, бюджет, ресурсы).
   3. Утверждение/отклонение CCB (включая заказчика).
   4. Обновление плана/документов при утверждении.
3. **Критика гибкости:**
   * *Аргументы против:* Не адаптируется к изменениям рынка/требований, долгая обратная связь, высокий риск провала при изначальных ошибках.
   * *Аргументы за:* Стабильность, предсказуемость для заказчика и команды, четкость ролей и процессов, удобство для аудита и отчетности.

**V. Сравнение с другими методологиями:**

1. **Waterfall vs Agile - критерии выбора:**
   * **Выбрать Waterfall:** Четкие/стабильные требования, жесткие сроки/бюджет (фиксированный контракт), строгая регуляторика, низкая толерантность к риску, большие команды/подрядчики.
   * **Выбрать Agile:** Неясные/меняющиеся требования, важна скорость вывода продукта/обратная связь, гибкость приоритетов, инновационный продукт, вовлеченный заказчик.
2. **Waterfall vs V-Model:** V-Model - *развитие Waterfall* для ПО. Ключевое отличие - **раннее тестирование:** Каждой фазе разработки (требования -> проектирование -> код) соответствует *заранее спланированная* фаза тестирования (приемочные тесты -> интеграционные тесты -> юнит-тесты). Тестирование встроено в процесс.
3. **Интеграция Agile в Waterfall (Гибриды):** Да, кроме Water-Scrum-Fall, примеры:
   * **Этапное прототипирование:** На этапах "Требования" или "Проектирование" создаются и валидируются прототипы с заказчиком (Agile-подход).
   * **Agile внутри этапа:** Как в Water-Scrum-Fall, или использование Kanban для управления задачами внутри этапа "Реализация".

**VI. Кейсы и примеры:**

1. **Успешные кейсы:**
   * **Строительство:** Любой крупный мост, туннель, небоскреб (например, Бурдж-Халифа) - требуется полный план до начала работ.
   * **Аэрокосмос:** Программа "Аполлон" (NASA), разработка новых моделей самолетов (Boeing, Airbus) - безопасность и регуляторика критичны.
   * **Госпроекты:** Создание систем электронного голосования (требует сертификации), крупная ИТ-инфраструктура (миграция).
2. **Провалы (часто из-за не Waterfall, а его применения):**
   * **Denver Airport Baggage System (1995):** Катастрофические сбои, многократное превышение бюджета. Причины: сложность недооценена, требования менялись, интеграция проблемна.
   * **UK NHS IT Programme (2002-2011):** Потрачено ~£10 млрд, закрыто как неудачное. Причины: огромный масштаб, негибкость к изменениям, плохое управление.
3. **Крупные корпорации:**
   * **NASA:** Использует строгие модификации Waterfall (часто с элементами V-Model) для миссий (например, марсоходы) из-за критичности безопасности, уникальности, долгих циклов. Процессы документированы до мелочей.
   * **Boeing:** Применяет Waterfall/V-Model для разработки авионики и систем управления самолетов (сертификация FAA). Разработка физических компонентов также строго последовательна.

**Методологии Agile**

**I. Общие вопросы:**

1. **Agile простыми словами:** Это философия и набор методов для гибкой разработки. Вместо одного "большого скачка" (как Waterfall), работа разбивается на маленькие кусочки (итерации/спринты 1-4 недели). В конце каждой итерации вы получаете *рабочий кусочек продукта*, показываете его заказчику, получаете обратную связь и *сразу корректируете* план следующих шагов. Цель - быстро реагировать на изменения и раньше давать ценность.
   * **4 ценности (Манифест Agile):**
     + Люди и взаимодействие важнее процессов и инструментов.
     + Работающий продукт важнее исчерпывающей документации.
     + Сотрудничество с заказчиком важнее согласования условий контракта.
     + Готовность к изменениям важнее следования первоначальному плану.
   * **12 принципов:** Главные - удовлетворение клиента через ранние поставки, приветствие изменений даже на поздних стадиях, ежедневное сотрудничество, мотивированные профессионалы, работающий продукт как мера прогресса.
2. **Разница Agile, Scrum, Kanban:**
   * **Agile:** Философия, набор принципов и ценностей (как "демократия").
   * **Scrum:** Конкретный *фреймворк* для реализации Agile. Имеет жесткие роли (Scrum Master, Product Owner, Команда), артефакты (Бэклог, Спринт) и события (Планирование, Дейли, Обзор, Ретроспектива). Работает итерациями (спринтами).
   * **Kanban:** *Метод* визуализации и управления потоком работ (часто на доске). Фокусируется на непрерывном потоке задач, ограничении "работы в прогрессе" (WIP), отсутствии фиксированных итераций. Более гибкий в процессах, чем Scrum.
3. **Другие фреймворки Agile:**
   * **Extreme Programming (XP):** Фокус на технических практиках (парное программирование, TDD, непрерывная интеграция), короткие циклы обратной связи.
   * **Feature-Driven Development (FDD):** Акцент на проектировании и построении модели, разработка по функциональностям (фичам).
   * **Crystal:** Семейство методологий (разного "веса"), адаптирующихся под размер и критичность проекта.
   * **SAFe (Scaled Agile Framework):** Фреймворк для масштабирования Agile на большие предприятия (множество команд).
   * **LeSS (Large Scale Scrum):** Другой подход к масштабированию Scrum.

**II. Практическое применение:**

1. **Внедрение Agile с нуля (пошагово):**
   1. **Обучение и осознание:** Обучение команды и стейкхолдеров принципам Agile/Scrum. Обсуждение "зачем нам это".
   2. **Определение пилотного проекта:** Выберите небольшой, не критичный проект для старта.
   3. **Формирование команды:** Определите Scrum Master, Product Owner (Владелец продукта), команду разработки (5-9 человек, кросс-функциональная).
   4. **Создание Product Backlog:** PO формирует первоначальный бэклог продукта (список всего, что нужно сделать).
   5. **Проведение первого планирования спринта:** Команда выбирает задачи из бэклога на первый спринт (1-4 недели), декомпозирует их.
   6. **Запуск спринта:** Начинается работа. Проводятся Daily Stand-ups.
   7. **Проведение Обзора спринта (Sprint Review):** Показ инкремента продукта заказчику/стейкхолдерам, сбор обратной связи.
   8. **Проведение Ретроспективы спринта:** Обсуждение "как нам улучшить процесс" внутри команды.
   9. **Итерация и адаптация:** Повторяйте циклы (п.5-8), постоянно улучшая процесс на основе ретроспектив. Постепенно масштабируйте.
2. **Инструменты:**
   1. **Jira:** Самый мощный и популярный для Scrum/Kanban. Гибкая настройка рабочих процессов, бэклогов, спринтов, отчетов (velocity, burn-down). Подходит для команд любого размера.
   2. **Trello:** Простой, визуальный (доски Kanban). Идеален для маленьких команд, нетехнических проектов или старта. Легко освоить.
   3. **Notion:** Универсальный инструмент для документации, бэклогов, досок Kanban. Гибкий, но требует настройки под Agile. Лучше для смешанных задач (Agile + документация).
3. **Эффективные Daily Stand-ups (Дейлики):**
   1. **Формат:** 15 минут максимум, стоя (чтобы не затягивать). Каждый участник отвечает на 3 вопроса:
      1. Что я сделал вчера для целей спринта?
      2. Что я планирую сделать сегодня?
      3. Какие препятствия мешают мне?
   2. **Ключ к успеху:** Фокус на прогрессе к *цели спринта*, а не на деталях. *Не решать проблемы* на дейлике! Scrum Master фиксирует препятствия и помогает устранить их *после* встречи. Пунктуальность.

**III. Agile в разных сферах:**

1. **Маркетинг:**
   * **Как:** Планирование кампаний короткими циклами (спринтами). Бэклог = список маркетинговых активностей (посты, email, реклама, исследования). Обзоры = анализ метрик (охват, конверсия), адаптация плана. Пример: Команда запускает A/B тест лендинга за спринт, анализирует результаты, решает что масштабировать.
   * **Кейс:** Spotify использует Agile-сквады (включая маркетинг) для быстрого запуска и тестирования новых функций и кампаний.
2. **Нетехнические проекты:**
   * **Да!** (HR, Event-менеджмент, Юристы, Продажи). **Как:**
     + Разбейте цель (нанять 5 специалистов, провести конференцию) на задачи.
     + Создайте бэклог (кандидаты->интервью->офферы; площадка->спикеры->реклама).
     + Работайте короткими циклами, фокусируясь на приоритетах.
     + Используйте доски (Trello, Jira) для визуализации потока.
     + Регулярно обсуждайте прогресс и адаптируйте план.
3. **Образование:**
   * **Как:** Разбивка курса на модули/темы (спринты). Бэклог = учебные цели и материалы. "Инкремент" = освоенные навыки за модуль. Обратная связь = тесты, обсуждения, ретроспективы курса. Персонализация траекторий обучения. Студенты сами планируют часть работы (в рамках бэклога).

**IV. Проблемы и решения:**

1. **Основные ошибки при переходе:**
   * Отсутствие поддержки руководства ("Agile только для команд").
   * Формальное внедрение ритуалов без понимания ценностей ("Agile-театр").
   * Неготовность PO активно работать с бэклогом и принимать решения.
   * Недостаток обучения команды.
   * Игнорирование технических практик (автотесты, CI/CD), ведущее к "водопаду внутри спринта".
   * Отсутствие реальных ретроспектив и улучшений.
2. **Мотивация сопротивляющейся команды:**
   * **Объясните "Зачем?":** Как Agile решит *их* текущие боли (хаос, переработки, непонятные приоритеты).
   * **Вовлекайте в процесс:** Дайте команде право голоса в выборе практик и улучшений (на ретроспективах).
   * **Начните с малого:** Пилотный проект, короткие спринты.
   * **Обеспечьте поддержку:** Компетентный Scrum Master, обучение.
   * **Празднуйте маленькие победы:** Признавайте прогресс, достижение целей спринта.
   * **Будьте терпеливы:** Изменение мышления требует времени.
3. **Постоянно срываются спринты:**
   * **Анализируйте причины (на Ретро):**
     + Переоценка возможностей? -> Улучшайте планирование (учет прошлого velocity, декомпозиция задач).
     + Внешние помехи/меняющиеся приоритеты? -> Защищайте команду от внешнего шума (Scrum Master), стабильные приоритеты в спринте.
     + Технический долг/плохое качество? -> Внедряйте технические практики (TDD, рефакторинг), выделяйте время на долг.
     + Непонятные/большие задачи? -> Декомпозируйте до размера < 1-2 дня, уточняйте критерии готовности (DoD).
   * **Действия:** Корректируйте объем планирования на следующий спринт (брать меньше), фокусируйтесь на завершении задач, а не на количестве.

**V. Артефакты и церемонии:**

1. **Эффективная Ретроспектива:**
   * **Цель:** Постоянное улучшение процесса команды.
   * **Форматы:**
     + **Start/Stop/Continue:** Что начать делать? Что прекратить? Что продолжить?
     + **Mad/Sad/Glad:** Что злит/огорчает/радует?
     + **4L (Liked, Learned, Lacked, Longed For):** Что понравилось? Чему научились? Чего не хватило? О чем мечтаем?
     + **Ретро на основе данных:** Анализ метрик (velocity, lead time, кол-во багов).
   * **Ключ:** Создать безопасную атмосферу, фокусироваться на решениях (хотя бы 1-2 действия на следующй спринт), владелец у действия.
2. **Эффективный Product Backlog:**
   * **Принципы:** Упорядочен по приоритету (PO), детализирован "сверху вниз" (верх - четко, низ - размыто), содержит *ценность* для пользователя/бизнеса.
   * **Как составлять:**
     1. Включайте все идеи/требования.
     2. Четко формулируйте элементы (User Stories: "Как [Роль], я хочу [Функция], чтобы [Ценность]").
     3. Оценивайте относительную сложность (Story Points).
     4. Регулярно рефайните (уточняйте, дробите, переоценивайте) вместе с командой.
     5. Ярко расставляйте приоритеты (ценность, стоимость, риск).
3. **Важные метрики:**
   * **Velocity (Скорость):** Среднее кол-во Story Points, завершенных командой за спринт. **Как использовать:** *Прогнозировать* объем работы на будущие спринты. *Не* сравнивать команды! Смотреть *тренд* (стабильность/рост).
   * **Lead Time (Время выполнения):** Время от появления задачи в бэклоге до её полного завершения. **Как использовать:** Измерять *общую эффективность* потока, выявлять узкие места (длинный LT -> проблемы). Стремиться уменьшать.
   * **Cycle Time (Время цикла):** Время от начала работы над задачей до её завершения. **Как использовать:** Анализировать эффективность *непосредственной работы* над задачами.
   * **Burn-down/Burn-up Charts:** Графики прогресса в спринте/проекте. Показывают, успевает ли команда.

**VI. Сравнение с другими методологиями:**

1. **Agile + DevOps:**
   * **Можно ли?** Не можно, а *нужно*! Они идеально дополняют друг друга.
   * **Как:** Agile фокусируется на *разработке* (быстрая поставка ценности, гибкость). DevOps фокусируется на *поставке и эксплуатации* (автоматизация сборки, тестирования, развертывания, мониторинг). Вместе:
     + Agile команда пишет код маленькими частями.
     + DevOps практики (CI/CD) позволяют *автоматически* и *быстро* интегрировать, тестировать и развертывать этот код.
     + Результат: Короткие циклы обратной связи от пользователя -> Быстрая адаптация (Agile) + Быстрая и надежная поставка (DevOps).
   * **Культура:** И Agile, и DevOps требуют культуры сотрудничества, автоматизации, непрерывного улучшения.
2. **Где Agile не работает (или сложно):**
   * **Жестко регулируемые отрасли:** Финансы (особенно банкинг), здравоохранение, АЭС (где требуется полная документация *до* разработки, формальные процедуры валидации - сложно совместить с гибкостью Agile).
   * **Проекты с фиксированными контрактами (цена/сроки):** Гибкость Agile противоречит фиксированным обязательствам перед заказчиком.
   * **Критичные для безопасности системы (частично):** Требуют тщательного предварительного анализа и валидации, что замедляет циклы. Гибридные подходы (Agile на некритичных частях) возможны.
   * **Отсутствие вовлеченного заказчика/PO:** Agile требует постоянной обратной связи и принятия решений по приоритетам.
   * **Команды без дисциплины и самоорганизации:** Agile полагается на мотивацию и ответственность членов команды.
   * **Очень большие команды без фреймворка масштабирования:** Чистый Scrum/Kanban плохо масштабируются на 100+ человек без SAFe, LeSS и т.д.