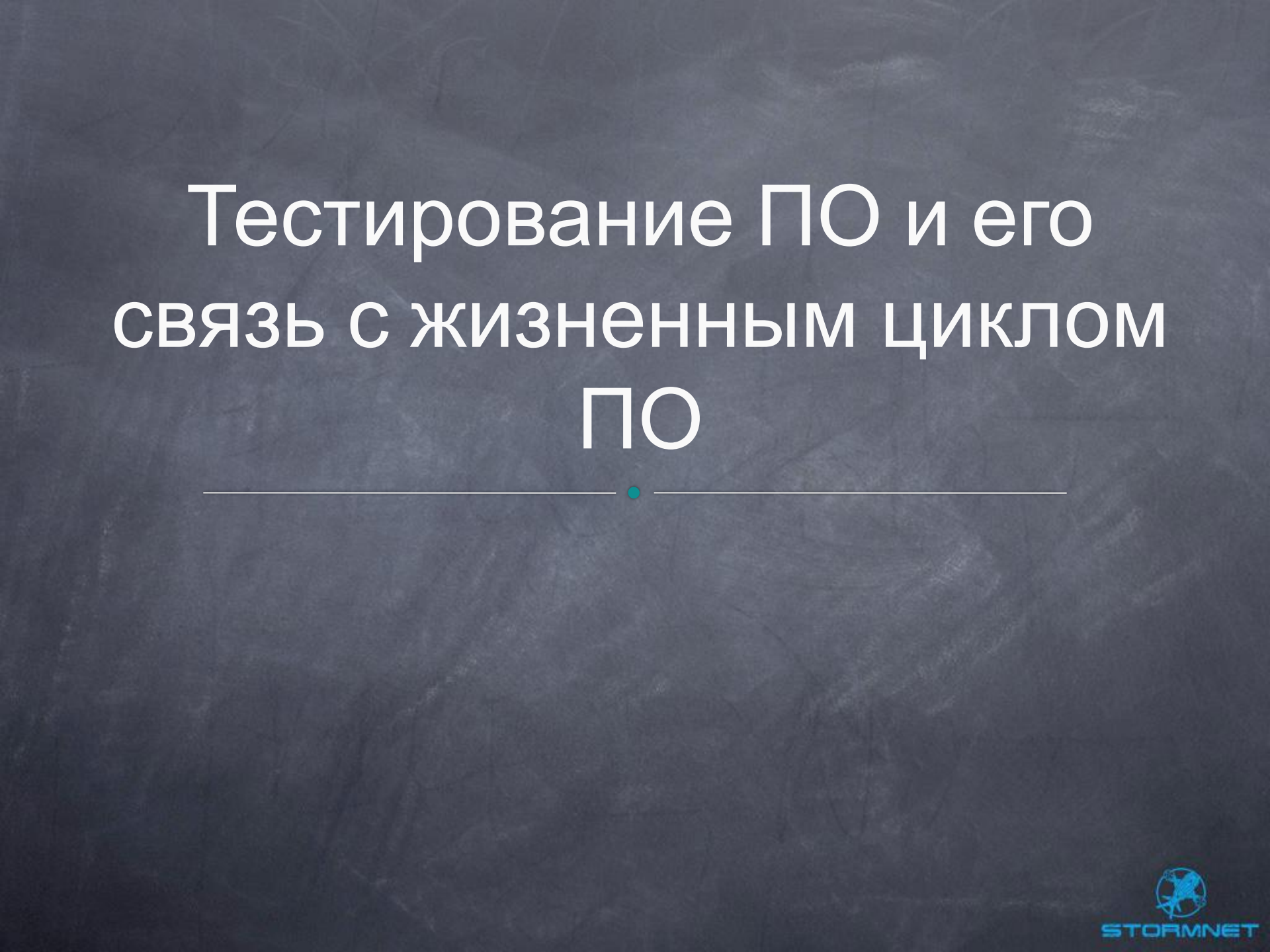
Тестирование ПО и его связь с жизненным циклом ПО



Цикл разработки ПО

- **Цикл (процесс) разработки ПО (software development life cycle)** — это путь от идеи до поддержки готового продукта. Чем более отлажены каждая из стадий цикла и координация между ними, тем эффективнее работает компания, тем выше качество и тем счастливее пользователи/клиенты.
- Сегодня мы поговорим о моделях цикла разработки ПО, называемых "Waterfall" ("Водопад"), и Agile («Гибкая»), которые используются в подавляющем большинстве компаний.

Жизненный цикл ПО

- Формирование и анализ требований
- Проектирование
- Реализация
- Тестирование продукта
- Внедрение
- Эксплуатация и сопровождение

Когда начать тестировать?

Давайте попробуем определить

STORMNET

Формирование и анализ требований

- Тестирование требований на вменяемость/ осуществимость
- Тестирование функциональной спецификации
- Создание предварительных тестовых сценариев на основе требований

Проектирование

- Тестирование документов, описывающих архитектуру (product architecture) и дизайн (product design)
- Тестирование плана проекта

Реализация

- Тестирования кода и модульное тестирование
- Тестирование результатов итерации разработки
- Регрессионное тестирование

Тестирование продукта

- Тестирование готового приложения и приемочное тестирование

Внедрение

- Тестирование совместимости продукта с существующей инфраструктурой заказчика (типы серверов, окружение и т. д.)
- Сбор и обработка обратной связи о приложении от пользователей

Эксплуатация и сопровождение

- Обработка отчетов об ошибках от пользователей во время эксплуатации
- Верификация исправления ошибок
- Обработка запросов новой функциональности
- Тестирование новой функциональности

Когда прекращать тестирование?

Давайте попробуем определить

STORMNET

Эвристики

- Эвристики – это быстрые, недорогие способы *решения проблемы* или *принятия решения*

Эвристики

- **Эвристика «Время вышло!»**. Для многих специалистов по тестированию это наиболее распространенная эвристика: мы останавливаем тестирование, когда заканчивается выделенное на него время.

Эвристики

- **Эвристика пиньяты** (The Piñata Heuristic). Мы прекращаем ломать программу, когда начинают выпадать конфеты – мы останавливаем тестирование, когда видим первую достаточно серьезную проблему.

- Пиньята (исп. Piñata) — мексиканская по происхождению полая игрушка довольно крупных размеров, изготовленная из папье-маше или лёгкой обёрточной бумаги с орнаментом и украшениями. Своей формой пиньята воспроизводит фигуры животных (обычно лошадей) или геометрические фигуры, которые наполняются различными угощениями или сюрпризами для детей (конфеты, хлопушки, игрушки, конфетти, орехи и т. п.)

Эвристики

- **Эвристика «мертвой лошади»** (The Dead Horse Heuristic). В программе слишком много ошибок, так что продолжение тестирования не имеет смысла. Мы знаем, что все изменится настолько, что сведет на нет результаты текущего тестирования.

Эвристики

- **Эвристика «Задание выполнено»** (The Mission Accomplished Heuristic). Мы останавливаем тестирование, когда найдены ответы на все поставленные вопросы.

Эвристики

- **Эвристика «Отмена задания»** (The Mission Revoked Heuristic). Наш клиент сказал нам: «пожалуйста, прекратите тестирование». Это может произойти по причине перерасхода бюджета, или вследствие отмены проекта, и по любой другой причине. Какова бы ни была причина, нам поручили остановить тестирование. (На самом деле эвристика «Время вышло!» может быть частным случаем более общей «Отмены задания», в том случае, если предпочтительнее, чтобы не мы сами, а заказчик принял решение о том, что время вышло.)

Эвристики

- **Эвристика «Я зашел в тупик!»** (The I Feel Stuck! Heuristic). По какой бы то ни было причине мы останавливаемся, поскольку обнаруживаем некое препятствие. У нас нет информации, которая нам требуется (например, многие люди заявляют, что не могут тестировать без достаточного количества спецификаций). Имеется блокирующая ошибка, и таким образом мы не можем перейти в ту область продукта, которую необходимо протестировать, у нас нет необходимого оборудования или инструментария, у команды нет квалификации, требуемой для выполнения некоторых специальных тестов

Эвристики

- **Эвристика «освежающей паузы»** (The Pause That Refreshes Heuristic). Вместо прекращения тестирования мы приостанавливаем его на некоторое время. Мы можем остановить тестирование и сделать перерыв, когда мы устали, когда нам стало скучно или пропало вдохновение. Мы можем сделать паузу на то, чтобы выполнить некоторые исследования, разработать планы, поразмыслить над тем, что мы делали в прошлом и понять, что делать дальше. Идея заключается в том, что нам требуется определенный перерыв, после которого мы сможем вернуться к продукту со свежим взглядом или свежими мыслями.

Эвристики

- **Эвристика «Отсутствие продвижения»** (The Flatline Heuristic). Что бы мы ни делали, мы получаем тот же самый результат. Это может происходить в случае, когда программа падает определенным способом или перестает отвечать, но также мы можем не продвигаться, когда программа в основном ведет себя стабильно: "выглядит хорошо!"

Эвристики

- **Эвристика «Привычного завершения»** (The Customary Conclusion Heuristic). Мы останавливаем тестирование тогда, когда мы обычно останавливаем тестирование. Имеется протокол, задающий определенное количество идей для тестирования, или тест-кейсов, или циклов тестирования, или как вариант – имеется определенный объем работ по тестированию, который мы выполняем и после этого останавливаемся. Agile-команды, например, часто применяют такой подход: «когда выполнены все приемочные тесты, мы знаем, что продукт готов к поставке». Эвальд Руденриджс (Ewald Roodenrijs) приводит пример этой эвристики в статье «Когда прекращать тестирование». Он говорит, что он останавливается, «когда выполнено определенное количество тестовых циклов, включая регрессионное тестирование».

Эвристики

- **Больше нет интересных вопросов** (No more interesting questions). В этот момент мы решаем, что не осталось вопросов, ответы на которые были бы достаточно ценными, чтобы оправдать стоимость продолжения тестирования, и поэтому мы останавливаемся. Эта эвристика используется в основном как дополнение к другим эвристикам, помогая принять решение о том, есть ли какие-то вопросы или риски, которые отменяют действие этих эвристик. Кроме того, если одна эвристика советует нам прекратить тестирование, следует проверить, нет ли интересных вопросов или серьезных рисков в других областях, и если они есть, то мы скорее продолжим тестирование, чем остановимся.

Эвристики

- **Эвристика уклонения/безразличия** (The Avoidance/Indifference Heuristic). Иногда людей не интересует дополнительная информация, либо они не хотят знать, что происходит в программе. Тестируемое приложение может быть первой версией, которую, как мы знаем, скоро заменят. Некоторые люди прекращают тестирование по причине лени, злого умысла или отсутствия мотивации. Иногда бизнес-критичность выпуска нового релиза настолько высока, что никакая мыслимая проблема не остановит выход программы, и поэтому никакие новые результаты тестирования не будут иметь значения.

Методологии разработки

- «Колхоз» против «Процесса»

STORMNET

Основная задача методологий

- Быстрота выполнения работ и четкая координация команд
- Качественное исполнение и контроль качества
- Сокращение издержек

Отличия от «Кустарного производства»

- Большой объем параллельных проектов
- Большая предсказуемость объема работ и результата

Методология описывает

- Роли
- Активности
- Артефакты
- Фазы и критерии их начала/окончания

Типичные роли

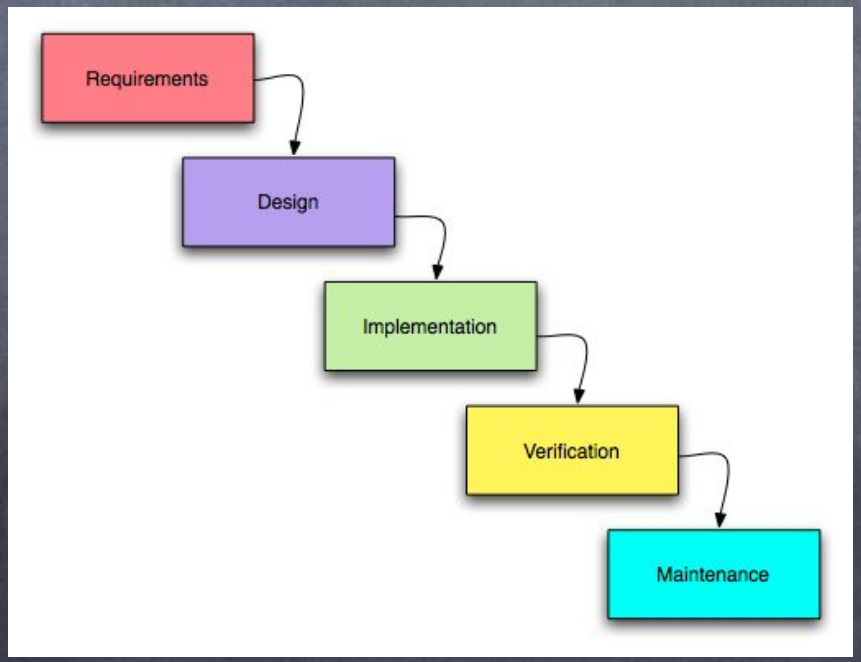
- Руководитель проекта (Project Manager)
- Архитектор (Architect/Designer)
- Бизнес-аналитик (Business Analyst)
- Разработчик (Developer)
- Тестировщик (Test Engineer/Tester)

Waterfall

- **Каскадная модель** (англ. waterfall model) — модель процесса разработки программного обеспечения, в которой процесс разработки выглядит как поток, последовательно проходящий фазы анализа требований, проектирования, реализации, тестирования, интеграции и поддержки

Waterfall

- Переход от одной фазы к другой происходит только после полного и успешного завершения предыдущей



Waterfall

Следуя каскадной модели, разработчик переходит от одной стадии к другой строго последовательно.

Сначала полностью завершается этап «определение требований», в результате чего получается список требований к ПО.

После того как требования полностью определены, происходит переход к проектированию, в ходе которого создаются документы, подробно описывающие для программистов способ и план реализации указанных требований.

После того как проектирование полностью выполнено, программистами выполняется реализация полученного проекта.

На следующей стадии процесса происходит интеграция отдельных компонентов, разрабатываемых различными командами программистов.

После того как реализация и интеграция завершены, производится тестирование и отладка продукта; на этой стадии устраняются все недочёты, появившиеся на предыдущих стадиях разработки. После этого программный продукт внедряется и обеспечивается его поддержка — внесение новой функциональности и устранение ошибок.

Waterfall

Тем самым, каскадная модель подразумевает, что переход от одной фазы разработки к другой происходит только после полного и успешного завершения предыдущей фазы, и что переходов назад либо вперёд или перекрытия фаз — не происходит.

Agile

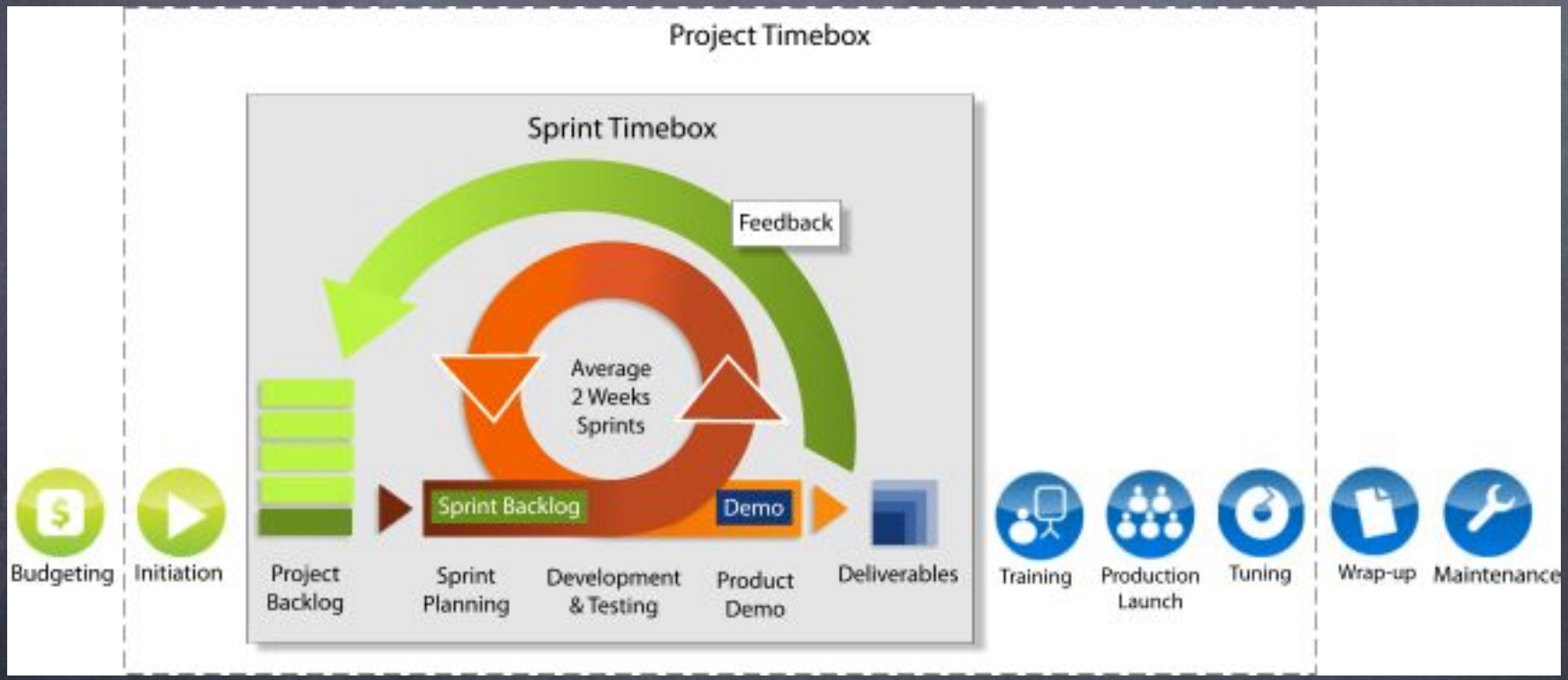
- **Гибкая методология разработки** (англ. Agile software development, agile-методы) — серия подходов к разработке программного обеспечения, ориентированных на использование итеративной разработки, динамическое формирование требований и обеспечение их реализации в результате постоянного взаимодействия внутри самоорганизующихся рабочих групп, состоящих из специалистов различного профиля.

Agile

- **Итеративный подход (англ. iteration, «повторение») в разработке программного обеспечения** — это выполнение работ параллельно с непрерывным анализом полученных результатов и корректировкой предыдущих этапов работы.
- Проект при этом подходе в каждой фазе развития проходит повторяющийся цикл:

Планирование — Реализация — Проверка — Оценка (англ. plan-do-check-act)

Agile



Преимущества итеративного подхода

- Снижение воздействия серьёзных рисков на ранних стадиях проекта, что ведет к минимизации затрат на их устранение
- Организация эффективной обратной связи проектной команды с потребителем (а также заказчиками) и создание продукта, реально отвечающего его потребностям
- Акцент усилий на наиболее важные и критичные направления проекта
- Непрерывное итеративное тестирование, позволяющее оценить успешность всего проекта в целом
- Раннее обнаружение конфликтов между требованиями, моделями и реализацией проекта
- Более равномерная загрузка участников проекта
- Эффективное использование накопленного опыта
- Реальная оценка текущего состояния проекта и, как следствие, большая уверенность заказчиков и непосредственных участников в его успешном завершении
- Затраты распределяются по всему проекту, а не группируются в его конце

Принципы Agile

- Agile — семейство процессов разработки, а не единственный подход в разработке программного обеспечения, и определяется Agile Manifesto (http://www.agilemanifesto.org/iso/ru/)

Agile-манифест

- *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*

- При этом: не отрицая важности того, что справа, мы всё-таки больше ценим то, что слева

Пояснение принципов Agile-манифеста

- Удовлетворение клиента за счёт ранней и бесперебойной поставки ценного программного обеспечения
- Приветствие изменений требований даже в конце разработки (это может повысить конкурентоспособность полученного продукта)
- Частая поставка рабочего программного обеспечения (каждый месяц или неделю или ещё чаще)
- Тесное, ежедневное общение заказчика с разработчиками на протяжении всего проекта
- Проектом занимаются мотивированные личности, которые обеспечены нужными условиями работы, поддержкой и доверием
- Рекомендуемый метод передачи информации — личный разговор (лицом к лицу)
- Работающее программное обеспечение — лучший измеритель прогресса
- Спонсоры, разработчики и пользователи должны иметь возможность поддерживать постоянный темп на неопределённый срок
- Постоянное внимание улучшению технического мастерства и удобному дизайну
- Простота — искусство не делать лишней работы
- Лучшие технические требования, дизайн и архитектура получаются у самоорганизованной команды
- Постоянная адаптация к изменяющимся обстоятельствам

Scrum

- **Scrum** (от англ. scrum «схватка») — методология управления проектами, активно применяющаяся при разработке информационных систем для гибкой разработки программного обеспечения. Scrum чётко делает акцент на качественном контроле процесса разработки.

- Scrum является одним из самых распространенных видов Agile

Scrum. Основные понятия

- **Скрам (Scrum)** – это набор принципов, на которых строится процесс разработки, позволяющий в жёстко фиксированные и небольшие по времени итерации, называемые спринтами (sprints), предоставлять конечному пользователю работающее ПО с новыми возможностями, для которых определён наибольший приоритет. Новые функции ПО, которые должны быть реализованы в очередном спринте определяются в начале спринта на этапе планирования и не могут изменяться на всём его протяжении. При этом строго фиксированная небольшая длительность спринта придаёт процессу разработки предсказуемость и гибкость.

Scrum. Основные понятия

- **Спринт (Sprint)** — итерация в скраме, в ходе которой создаётся функциональный рост программного обеспечения. Жёстко фиксирован по времени. Длительность одного спринта от 2 до 4 недель.

Scrum. Основные понятия

- **Резерв Проекта (Product backlog)** — это список требований к функциональности, упорядоченный по их степени важности, подлежащих реализации. Элементы этого списка называются «пожеланиями пользователя» (user story) или элементами резерва (backlog items).
- Резерв проекта открыт для редактирования для всех участников скрам процесса.

Scrum. Основные понятия

- **Резерв спринта (Sprint backlog)** — содержит функциональность, выбранную владельцем проекта из резерва проекта.
- Все функции разбиты по задачам, каждая из которых оценивается скрам-командой.
- Каждый день команда оценивает объем работы, который нужно проделать для завершения спринта.

Scrum. Основные понятия

- **История спринта (Sprint Story)** — Требуемая функциональность, которую добавляют в резерв, часто называют историей. Зачастую история имеет следующую структуру: *«Будучи пользователем ‹тип пользователя› я хочу сделать ‹действие›, чтобы получить ‹результат›»*.
- Такая структура удобна тем, что понятна как разработчикам так и заказчикам.
- История, которая исходит от пользователя, называется «Пожелание пользователя» (User Story)

Scrum. Роли

По методике Scrum в производственном процессе есть определенные роли, разбитые на 2 группы «свиней» и «кур». Эти названия были использованы из-за шутки:

Свинья идёт по дороге. Курица смотрит на нее и говорит: «А давай откроем ресторан!» Свинья смотрит на курицу и отвечает: «Хорошая идея, и как ты хочешь его назвать?» Курица думает и говорит: «Почему бы не назвать 'Яичница с беконом'?». «Так не пойдёт», — отвечает свинья, «ведь тогда мне придётся полностью посвятить себя проекту, а ты будешь вовлечена только частично.»

Scrum. Роли

- «Свиньи» – основные роли. Вовлечены в проект полностью
- «Куры» – второстепенные роли. Вовлечены в проект частично

Scrum. Основные роли («Свиньи»)

«Свиньи» полностью включены в проект и в скрам-процесс.

- **Скрам-мастер (ScrumMaster)** — проводит совещания (Scrum meetings) следит за соблюдением всех принципов скрам, разрешает противоречия и защищает команду от отвлекающих факторов. Данная роль не предполагает ничего иного, кроме корректного ведения скрам-процесса. Руководитель проекта скорее относится к владельцу проекта и не должен фигурировать в качестве скрам-мастера.
- **Владелец продукта (Product Owner)** — представляет интересы конечных пользователей и других заинтересованных в продукте сторон.
- **Скрам-команда (Scrum Team)** — кросс-функциональная команда разработчиков проекта, состоящая из специалистов разных профилей: тестировщиков, архитекторов, аналитиков, программистов и т. д. Команда является единственным полностью вовлечённым участником разработки и отвечает за результат как единое целое. Никто кроме команды не может вмешиваться в процесс разработки на протяжении спринта.

Scrum. Дополнительные роли («Куры»)

- Пользователи (Users)
- Клиенты, Продавцы (Stakeholders) — лица, которые инициируют проект и для кого проект будет приносить выгоду. Они вовлечены в скрам только во время обзорного совещания по спринту (Sprint Review).
- Управляющие (Managers) — люди, которые управляют персоналом.
- Эксперты-консультанты (Consulting Experts)

Scrum. Встречи (Meetings)

Планирование спринта (Sprint Planning Meeting) Происходит в начале новой итерации Спринта

- Из резерва проекта выбираются задачи, обязательства по выполнению которых за спринт принимает на себя команда
- На основе выбранных задач создается резерв спринта. Каждая задача оценивается в идеальных человеко-часах
- Решение задачи не должно занимать более 12 часов или одного дня. При необходимости задача разбивается на подзадачи
- Обсуждается и определяется, каким образом будет реализован этот объём работ
- Продолжительность совещания ограничена сверху 4-8 часами в зависимости от продолжительности итерации, опыта команды и т. п.
  - (первая часть совещания) Участвует владелец проекта и скрам команда: выбирают задачи из резерва продукта
  - (вторая часть совещания) Участвует только команда: обсуждают технические детали реализации, наполняют резерв спринта

Scrum. Встречи (Meetings)

Ежедневное совещание (Daily Scrum meeting):

- Начинается точно вовремя
- Все могут наблюдать, но только «свиньи» говорят
- Длится не более 15 минут
- Проводится в одном и том же месте в течение спринта.

В течение совещания каждый член команды отвечает на 3 вопроса:

- Что сделано с момента предыдущего ежедневного совещания?
- Что будет сделано с момента текущего совещания до следующего?
- Какие проблемы мешают достижению целей спринта? (Над решением этих проблем работает скрам мастер. Обычно это решение проходит за рамками ежедневного совещания и в составе лиц, непосредственно затронутых данным препятствием.)

Scrum. Встречи (Meetings)

Обзор итогов спринта (Sprint review meeting) Проводится после завершения спринта:

- Команда демонстрирует инкремент функциональности продукта всем заинтересованным лицам.
- Привлекается максимальное количество зрителей.
- Все члены команды участвуют в демонстрации (один человек на демонстрацию или каждый показывает, что сделал за спринт).
- Нельзя демонстрировать незавершенную функциональность.
- Ограничена четырьмя часами в зависимости от продолжительности итерации и инкремента продукта.

Scrum. Встречи (Meetings)

Ретроспективное совещание (Retrospective meeting)
Проводится после завершения спринта:

- Члены команды высказывают своё мнение о прошедшем спринте.
- Отвечают на два основных вопроса:
  - Что было сделано хорошо в прошедшем спринте?
  - Что надо улучшить в следующем спринте?
- Выполняют улучшение процесса разработки (решают вопросы и фиксируют удачные решения).
- Ограничена одним—тремя часами.