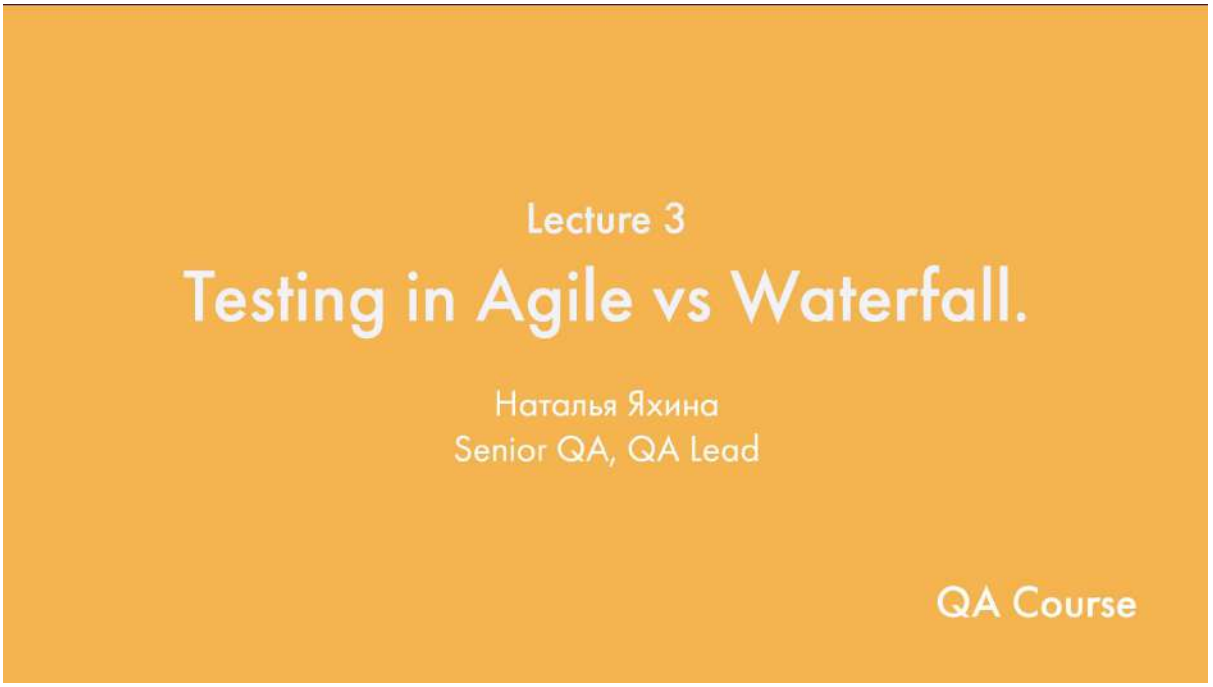


Тестирование по Agile и Waterfall. Анализ результатов тестирования. Репортинг.

Я - Наталья Яхина, Senior QA Lead в DataArt, также веду свой блог в Инстаграм про тестирование, если интересно, заходите: @qa_career_up. Сегодня у нас с вами заключительная лекция и поговорим про тестирование в различных методологиях, в частности Agile и Waterfall.

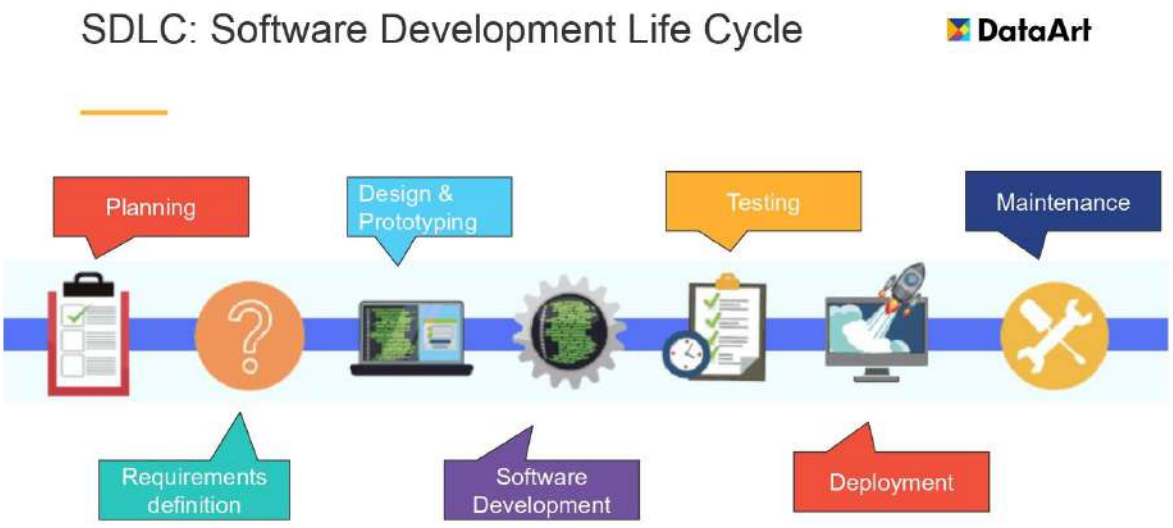
Небольшой план на сегодня:

В первой части мы поговорим про жизненный цикл программного обеспечения, методологии, которые существуют и непосредственно про тестирование, и его место в каждой из методологий. Вы уже знаете, кто такой тестировщик, как он тестирует, какими инструментами пользуются. И сегодня хочется ответить на вопрос: **когда** и **где** тестировщик тестирует в большом **процессе разработки ПО**.

Во второй части мы посмотрим, какие **результаты** получаем по итогам тестирования, какие метрики можно собирать и **зачем**, какой репорт (**отчёт**) можно составить, **кому** отправить, что мы таким образом получим.

Давайте, приступим к первой части, **тестирование Agile и Waterfall**. Надеюсь, для вас это знакомые слова они достаточно на слуху. Но я расскажу подробнее, сначала остановимся на понятии **жизненный цикл** программного обеспечения. Программа начинает жить тогда, когда появляется идея об этой программе, и заканчивает тогда, когда выводится из эксплуатации. Если мы ничего не разрабатываем, но программа живёт, у нас есть какие-то пользователи, можно считать что программа находится на стадии поддержки. Хочется провести аналогию с человеческим жизненным циклом, мы рождаемся, мы умираем. Но между этими событиями есть определенные этапы жизни: младенчество, детство, взрослость, старость.

Также и у программы: есть стадия **планирования**, т.е. непосредственная идея и видение того, как это будет, какие проблемы будет решать то или иное ПО. Следующая стадия - это **разработка и утверждение требований**, здесь уже более конкретно мы рассматриваем, что будет делать программа, как она это будет делать, какими средствами, какие ресурсы для этого нужны. Дальше будет **стадия дизайна и прототипирование** - это всё, что связано с архитектурой, с подготовительными работами, визуальным дизайном и так далее.

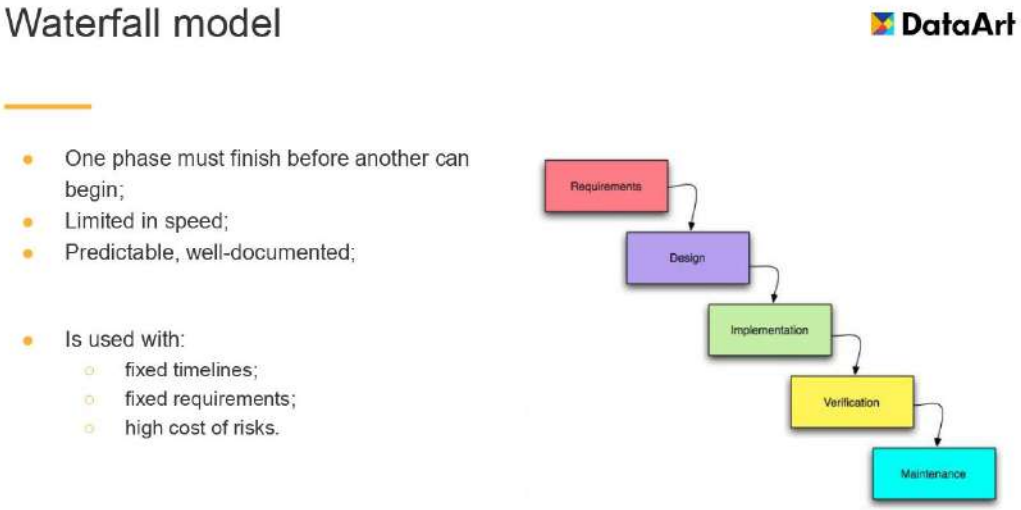


Затем мы переходим к **разработке продукта**. Здесь имеется в виду написание кода, затем стадия **тестирования**, затем **релиз**. И когда мы часть продукта или весь продукт передали в использование, мы переходим к стадии **поддержки**. **Вот 7 основных стадий в жизненном цикле ПО.**

Однако, если у человека всё последовательно, мы от рождения до старости проходим через все стадии последовательно, то у продуктов бывают разные вариации того, как компоновать, как переходить от одной стадии к другой, и как раз за это у нас будет отвечать **методология разработки ПО**.

Небольшой пазл для вас, за каждой картинкой скрывается определенная методология (предлагаю поставить на стоп и выполнить тест: запишите на листе, как вы думаете, какая модель зашифрована на каждой из картинок. ответы будут в конце). У нас их много различных, не получится рассмотреть каждую подробно, поэтому посмотрим самые популярные. Часто используется методология **каскадной модели разработки**, или **водопадная**. И так называемый **гибкий подход**. Гибкий подход - это семейство методологий, самые основные из них мы тоже рассмотрим. Хочется сделать акцент на том, что когда говорят про Agile (или про гибкую разработку) - это не конкретная методология - это семейство, много видов.

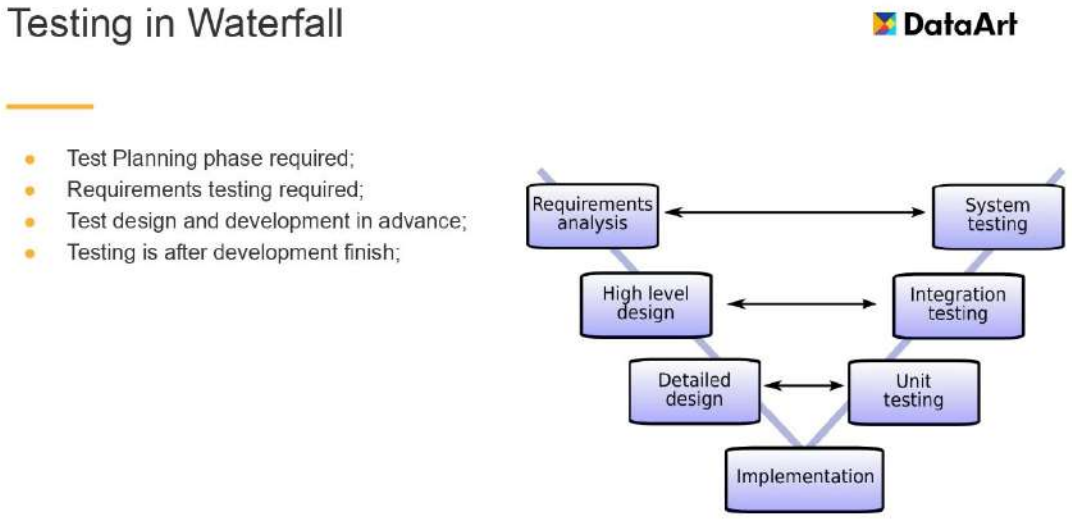
Waterfall.



Каскадная модель - здесь особенность: все стадии разработки идут **последовательно**, каждая **новая стадия начинается только по завершении предыдущей**. И нельзя возвращаться, здесь, как видите, нет стрелки назад. То есть как только мы обсудили, анализировали требования, утвердили, в идеале еще и протестировали (требования), переходим к стадии дизайна. Как только была создана архитектура, разработаны технические моменты, дизайн, если есть какой-то визуал, мы переходим к стадиям имплантации, т.е. написании кода.

Когда здесь вводят тестировщика? **Как можно раньше**, и на самом деле работа здесь найдётся на каждой из стадий для нас. Но непосредственно **тестирование проводится в определённой чёткий момент времени после разработки**. Т.е. выполнение тестов будет происходить в конкретный момент, где жёлтый квадратик (не раньше, и не параллельно). Особенность каскадной модели заключается в том, что всё должно быть готово, чтобы приступить к тестированию. Это несколько замедляет процесс, но в этом есть существенное преимущество, которое заключается в том, что мы двигаемся **последовательно**, нет никаких неожиданностей, всё достаточно предсказуемо. Мы ограничены в скорости, есть какие-то точные сроки, когда мы начнём и закончим разработку. Как правило, каскадная модель хорошо **задокументирована**, так как у нас есть определённая стадия, где мы разрабатываем требования, соответственно, мы не можем приступать к тестированию раньше, чем закончится имплементация. У нас есть время на то, чтобы **продумать тестовый сценарий, подготовить тестовые данные**, задокументировать работу, в идеале написать **тест план**. Всё движется последовательно и размеренно.

Когда используется Waterfall? Когда у нас четко **определенные сроки** на разработку продукта, когда есть зафиксированной **чёткие требования** к тому, что мы хотим получить на выходе, и когда **высока цена рисков**. Отсюда вытекает, что часто Waterfall используется в различных приложениях, связанных с жизнью, медициной, ракетостроение и так далее. То есть что-то тяжеловесное, размеренное и ответственное.



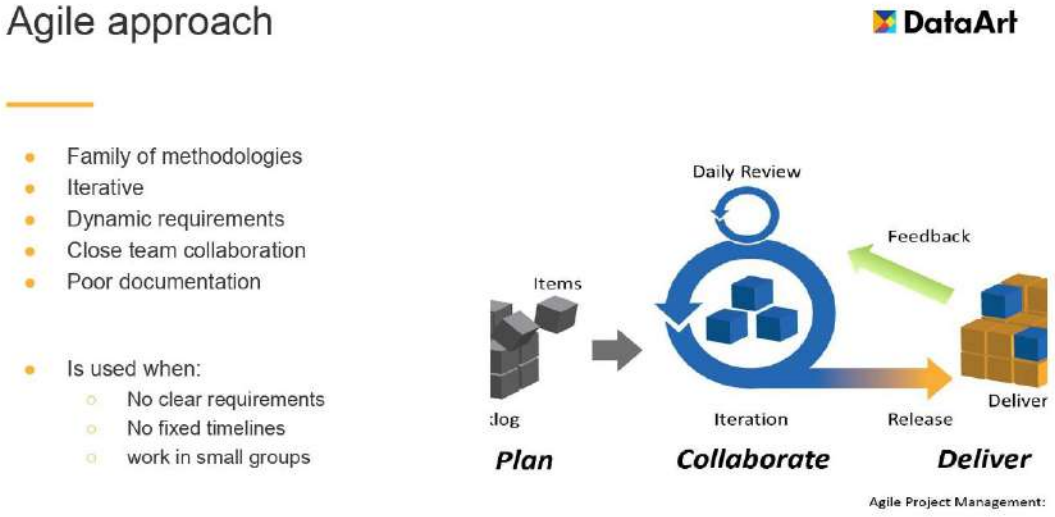
Также есть разновидность каскадной модели, так называемая V-model, она не сильно отличается от Waterfall, представлена в форме буквы V, на левой стороне все

стадии связанные с разработкой, на правой половине - стадии связанные с тестированием. И стрелки между стадиями показывают, что одна стадия будет влиять на другую. Например, анализ требований будет влиять на системное тестирование.

Что для тестировщиков интересно в плане тестирования в таких моделях? Есть определённая стадия планирования, пока мы ждем, пока будет закончена имплементация, у нас есть время на то, чтобы разработать тест план, чтобы спланировать активности, просчитать ресурсы и тд. И как правило это стадия реквайментов, она обязательна. Т.е. есть определенная стадия разработки и тестирования требований, и Waterfall подразумевает, что они не могут меняться после утверждения. Это ответственная задача, нужно максимально сфокусировано проверить требования, провести статическое тестирование. Разработка тестов точных или автоматизированных производится заранее, как говорили уже ранее, и начинается после разработки.

Agile.

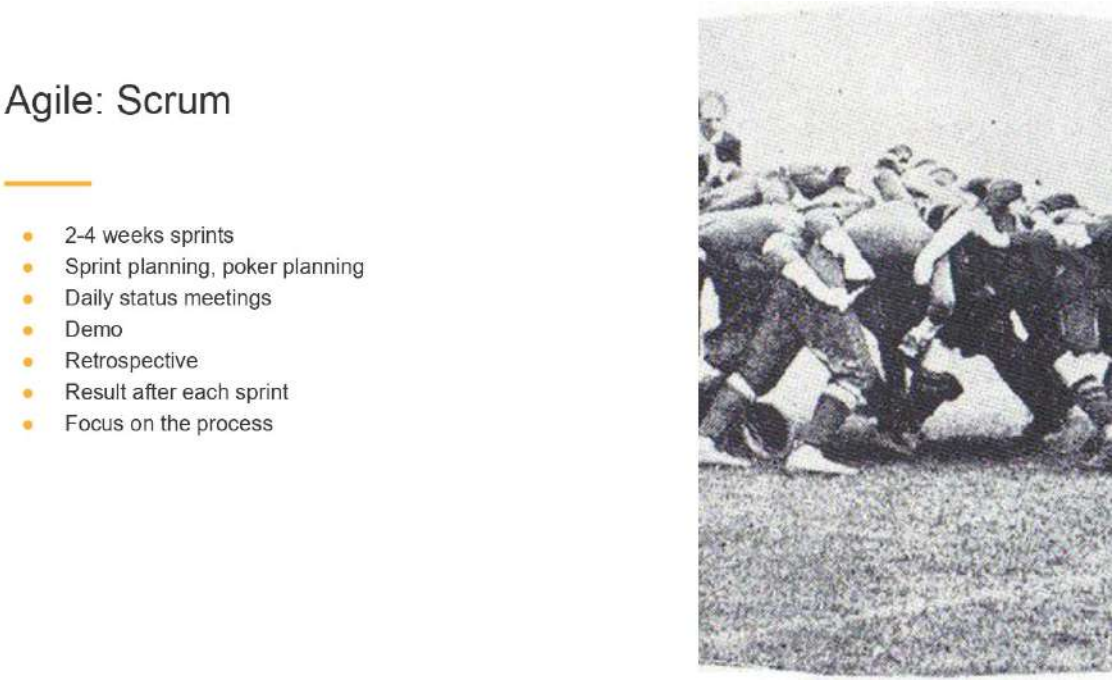
Большинство приложений, которыми мы с вами пользуемся разрабатывают именно с использованием гибких подходов. Agile - это семейство методологий, их объединяет то, что они все **итеративные**, они подразумевают динамичность требований. Требования могут меняться в процессе разработки, тестирования и вообще в любой момент времени, приходится из-за этого **много общаться** внутри команды. Потому что если **требования меняются**, то очень важно донести это до каждого члена команды, ничего не пропустить. Но всё равно, из-за того, что всё динамично происходит, такие изменения не будут задокументированы. Поэтому особое требование к участникам команды - это высокий soft skills, умение общаться, задавать правильные вопросы, умение находить информацию.



Когда гибкие подходы используются? Когда нет чётких требований. Когда нет конкретных сроков. И работа производится в небольших группах. Если это большой проект, то есть Agile коучи, люди которые обучают этим методологиям, они советуют избегать большую группу на несколько маленьких, тогда это будет более эффективно.

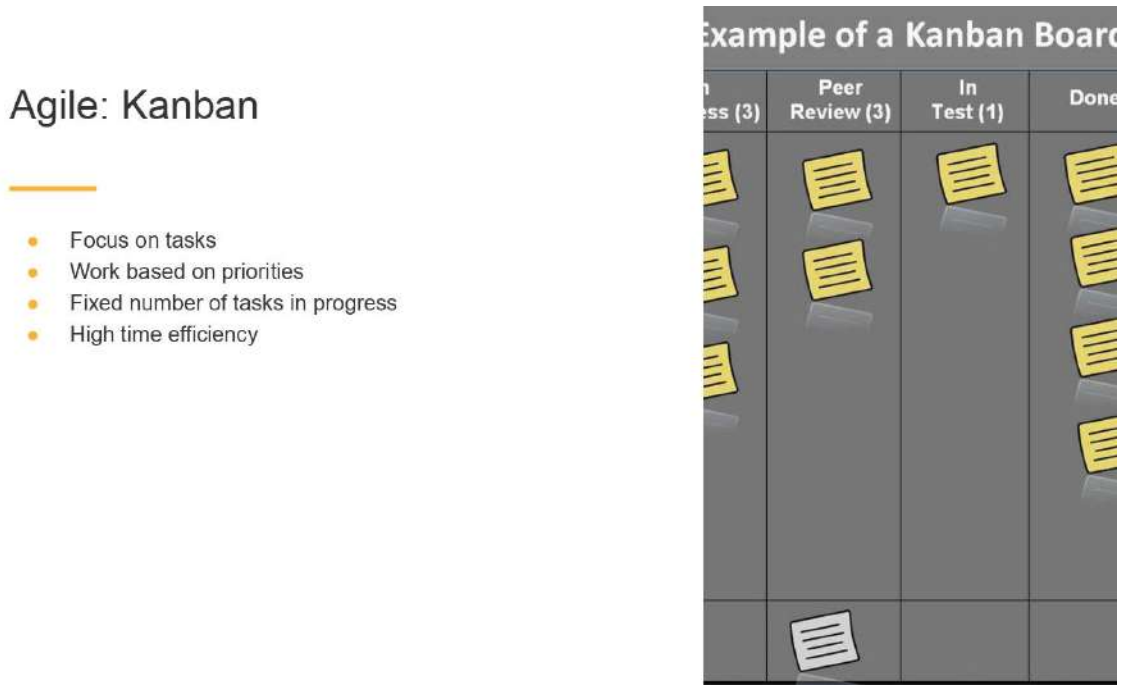
Давайте поподробнее посмотрим на под виды гибких методологий.

Scrum - выполнение определенных церемоний, высокий фокус на процесс, разработка по спринтам, каждый спринт длится 2-4 недели, в конце спринта предполагается: релиз, выпуск продукта или достигнута поставленная цель.



Спринты планируются заранее на специальных митингах. Проверяется bug log, обсуждается с командой, проставляются оценки, по итогам планирования формируется цель на ближайшее спринт. Предполагается, что по окончанию спринта все задачи будут выполнены с учётом тестирования. Бывает, что тестирование включено в текущей спринт. И бывает догоняющее тестирование, когда задачи на разработку ставятся на ближайший спринт, и предполагается, что тестироваться будут через спринт. Но обычно стараются делать всё более компактно. В идеале в конце каждого спринта проводится демо, где показывают всей команде или заказчику, стейкхолдерам, что было сделано, какие результаты достигнуты.

Ретроспектива - это тоже определённая церемония, предполагаемая в скраме, когда люди обсуждают, что было плохо в спринте, что прошло хорошо, и какие действия нужно предпринять для того, чтобы не повторить совершённых ошибок.



Следующий подвид методологии называется **Kanban**. В отличие от скрама, где фокус на процессе, в Kanban фокус на выполненную задачу. Вы наверняка слышали о Kanban board, он используется часто и в скраме. Это доска, где вывешиваются задачи, которые предстоит сделать, над которыми работают сейчас и которые уже сделаны. В канбане всегда работают над наиболее приоритетными задачами, если в ходе работы появляются ещё более приоритетные задачи, то предполагается бросить всё и переключиться. Здесь фиксированное количество задач, которые человек может выполнять в параллели, как правило, это одна задача. Над несколькими задачами **параллельная работа не приветствуются**, т.к. это ведёт к существенной деградации продуктивности. Самое главное преимущество канбана - это **время и эффективность**, используя его, мы сокращаем метрику time-to-market - время на поставку задачи пользователя.

Если в скраме мы должны ждать окончания спринта для итогового релиза, то в канбан мы отдаем продукт пользователю по готовности. Это удобно, если есть высокое время ожидания по поставки продуктов.

Рассмотрим ещё один подвид - **экстремальное программирование**. Метод экстремального программирования использует подход, который называется test-driven development - это разработка исходя из теста.

Agile: Extreme programming

- Test driven development
- Pair programming
- Review process
- No testers -> possible non functional problems

- Как правило в этом процессе участвуют сами разработчики, они изначально пишут тесты и под эти тесты пишут функциональный код.
- Используется парное программирование и ревью.
- Часто в таких методологиях избегают участия тестировщиков, что в свою очередь ведет к нефункциональным проблемам с продуктом.

В чистом виде вы не встретите экстремальное программирование в своей работе, но с ним делаются **гибридные подходы**.

Тестирование в гибкой методологии:

Testing in Agile

DataArt

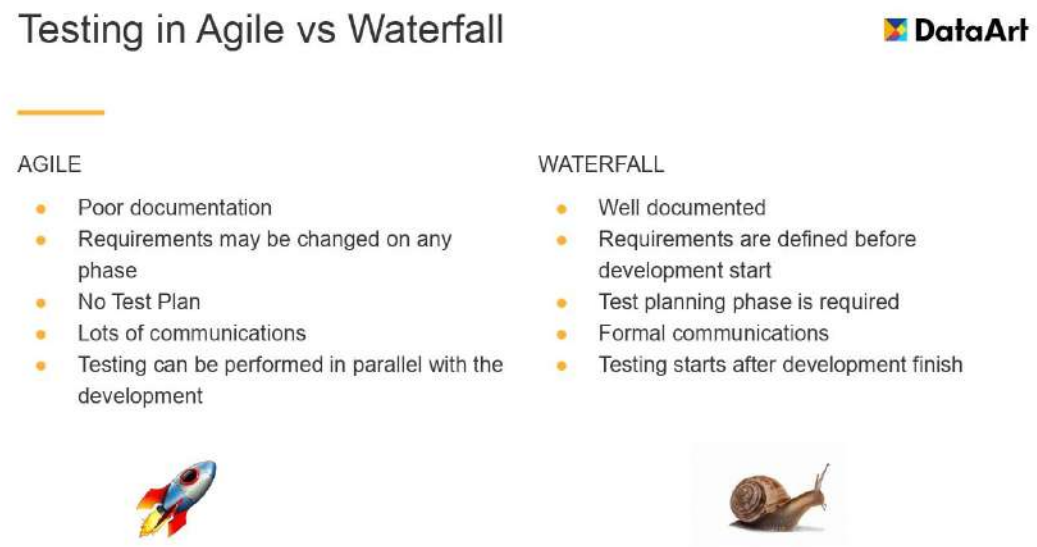
- Constant collaboration with development team
- Constant requirements clarification
- QA takes over BA role
- No detailed test plan
- Possible use of check lists instead of test cases
- Incremental testing

- постоянное взаимодействие тестировщика с командой разработки: выяснения требования, приоритезация;

- разъяснение в каких-то моментах, т.к. часто недостаточно документации;
- часто роль бизнес аналитика переходит к QA инженеру, опять же из-за отсутствия времени, недостаточной документации;
- постоянно приходится выяснять требования, даже если у вас есть чёткая задача в таск-трекере, всё равно приходится уточнять: актуальность, не изменилось ли что;
- в гибких проектах не всегда делается тест план, т.к. необязательное требование; иногда его пишут, если есть время и ресурсы, но чаще этот шаг просто пропускают;
- не пишут тест-кейсы, обходятся чек-листами, всё делается в экспресс формате, что с одной стороны очень удобно и быстро, с другой стороны вносит неопределённость, бардак в работу;
- чаще всего происходит инкрементальное тестирование. Мы берём тестирования части приложения или функционала, который готов.

В Waterfall мы ждём полного окончания стадии разработки, в гибких методологиях мы не ждём. Таким образом мы сокращаем цену ошибки, потому что чем раньше мы находим баги, тем дешевле их устранить. Потому что не появляются новые зависимости, нет каскада появления багов.

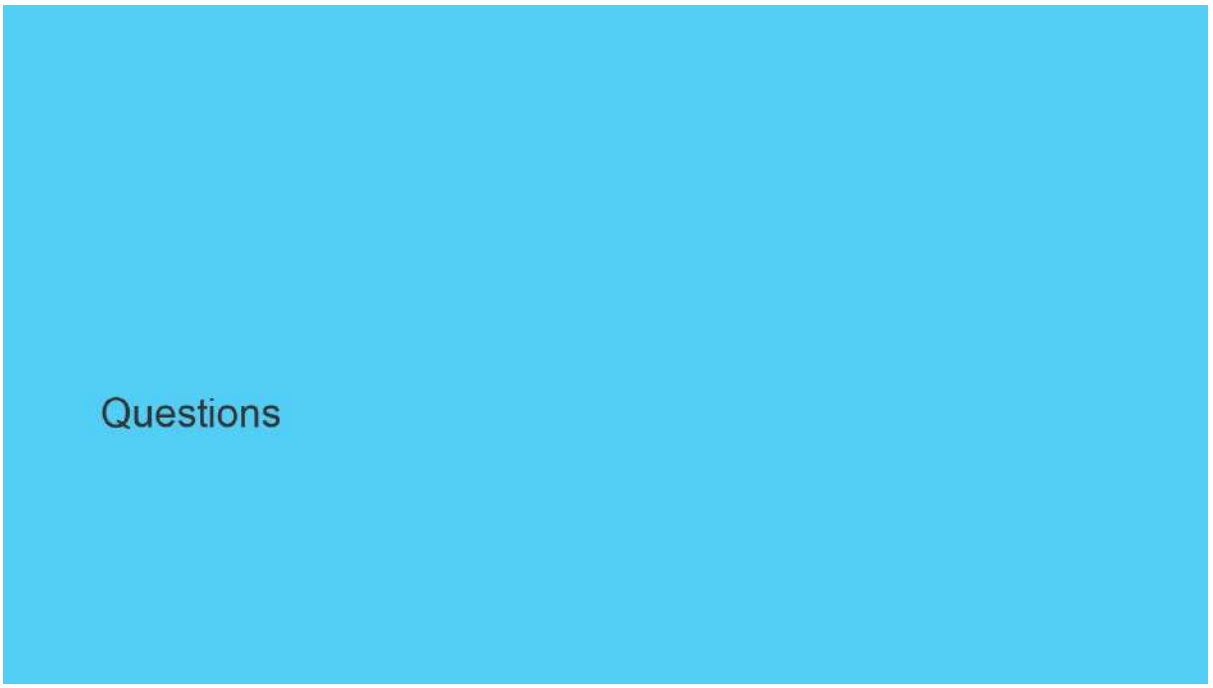
Посмотрим сводную таблицу:



Мы уже прошлись по основным отличиям, сравним их:

- Документация в гибких методология очень бедная, в Waterfall - это must have.
- В Agile требования меняются постоянно, нужно быть готовым к тому, придётся обновлять свои тесты. В Waterfall это маловероятно.
- В Agile нет стадии планирования, тест-планы редко пишутся. В Waterfall тест-план обязателен.
- В Agile будет много различных обсуждений и коммуникаций, в Waterfall тоже общаются, но будут более формальные коммуникаций, как правило, через e-mail с копиями на всех стейкхолдеров и заинтересованных лиц.

- Тестирование в Agile можно проводить параллельно с разработкой. В Waterfall будем ждать завершения стадии разработки перед тем, как переходить к тестированию.



Вопросы по первой части.

1. **Scrumban - как это может выглядеть?**

Гибких методологий в чистом виде вы не встретите, Scrumban - это самый популярный подвид, когда вы работаете по скраму, но при этом вы берёте практики приоритетные для канбан. Есть канбан борд, есть приоритеты, ограничено число задач в работе.

2. **В какой методологии больше всего находят ошибки тестировщики? Есть ли статистика?**

Честно говоря, не располагаю статистикой, но в гибких методологиях, они быстрее находятся, быстро фиксится, часто бывают регрессии (пофиксил одно, сломалось другое).

3. **Кто и в какой момент определяет методологию в проекте?**

Вам, как тестировщику, не придётся выбирать методологию, этим занимаются проект-менеджеры, стейкхолдеры, независимые консультанты. Бывают случаи, когда методологию на проекте могут поменять. Например, у меня на проекте работали по скраму, сейчас сделали чистый канбан, потому что для заказчика стало важно ещё быстрее получать фичи.

4. **В процентном соотношении какая методология используется в DataArt больше и почему?**

Снова именно по-моему опыту - это скрам в связке с канбаном. Но и по Waterfall у нас тоже есть определённые проекты.

5. **Кто ставит задачи в Канбан? Есть ли в Канбане баг репорт?**

Либо проект-менеджер, либо проект оунер, т.е. человек с видением бизнеса и приоритетов, потому что именно на это идёт упор в канбане, что важнее поставить пользователю в первую очередь. Да, в Канбане есть баг репорт, вы находите баги, заводите, и они также будут отображаться на доске.

6. **От предыдущих лекторов слышали нейтрально-негативные отзывы об Agile, где будет проще работать новичку?**

Нужно быть гибким и постоянно на связи, понимать, что бывают жуткие проекты, где вы тестируете и разрабатываете месяц, а к вам приходят стейкхолдеры и говорят: наши конкуренты выпустили такую фичу, мы тоже хотим её, поэтому ставим всё на холд и работаем над другим. Т.е. вы работали над чем-то интересным целый месяц, а сейчас это абсолютно не в приоритете, поэтому важен навык стрессоустойчивости. Лично думаю, что новичку будет работать одинаково вне зависимости от методологии.

7. **Как пишутся unit тесты в Waterfall?**

Unit тесты пишут разработчики. Даже, когда вы занимаетесь QA Automation тестированием, всё равно они на руках разработчиков. Поэтому будет правильнее сказать, что unit тесты включены в стадию разработки.

8. **Непонятно, как в Waterfall можно откладывать тесты до завершения стадии разработки.**

Да, это непросто. Но мы можем заранее проектировать тесты, продумывать сценарии, делать заглушки, делать моки. И отлаживать с заглушками вполне реально.

9. **Приведите практический пример: фокус на процессе, фокус на результате.**

Если мы говорим про скрам, где фокус на процессе, у нас много времени тратится на проведение церемоний и дейли митингов, вы в постоянном коннекте. В канбане фокус на результате, и большинство звонков опускается, идёт фокус на разработке задач. Задачи поставляются пользователям быстрее.

10. **Задачи в Канбан ставятся не на человека, а на команду. Может ли проект-менеджер ставить на конкретного сотрудника?**

Да, верно. Задача ставится на команду, потом распределяется на конкретных исполнителей. Есть лиды, которые распределяют задачи.

11. **Про Poker в чистом скраме почти 90% отзывов, что не занимаемся такой чепухой.**

У меня в последних проектах этим занимались, и очень хорошо работало. Когда небольшая команда, и все в контексте, все обладают определёнными техническими деталями. В скраме есть отличительная особенность, что более высокое требование к профессионализму. Т.е. если все в контексте и высокого уровня, не бывает проблем.

12. **В скраме одни звонки, когда работать?**

Конечно, все работают, но очень много митингов - это факт.