- Изначально задача находится в статусе **New**, разрабатываются требования, определяется кто задачу будет выполнять – переводят в статус **Asignee**

- Берётся задача в спринт, определяется для нее в какой из спринтов она будет включена (**Fix version**), переводится в статус **In progress**

**-** Разработчики занимаются написание функционала, когда они считают, что задача готова – переводят в статус **Review**, смерживают ветку

- Задача переходит в статус **Ready for testing**, соответственно задача готова к тестированию

- Тестировщик берёт задачу в работу – переходит в статус **In QA,** пишет тест-кейсы, формирует их в тест ран

**-** После того, как тестировщик его протестировал – задача переходит в статус **Ready.** Когда все задачи из данного спринта принимают статус **Ready –** можно приступать к регрессу

Затем начинается регресс, проверяющий, что весь функционал работающий ранее не сломался. Проходит регресс как побочного эффекта…если у нас какие-то баги были заведены ранее и у нас написаны автотесты мы проходим регресс старых багов….регресс новых багов уже идет во время основного регресса, когда все задачи вернули вам после правок.

Автоматизированы кейсы, у которых статусы **Blocker** и **Critical –** кейсы покрывают функционал критично важный для заказчика

Модель MO\_S\_C\_O\_W

* MO – must have, обязательная функциональность, которая должна проверяться
* Should have – уже чуть менее приоритетные кейсы
* Could – кейсы со статусами Medium и т.д.
* Would be – функционал, который хотелось бы, но не обязательно (Trivial)

Если в процессе регресса какие либо из тестов не прошли, упали – наш автоматизатор локально прогонял их по одиночке и разбирался в чем проблема…если же они снова падали – мы проходили их в ручную, пытались понять что не так (например, может тесты написано неправильно)…..

После этого мы создавали задачу, грубо говоря подразумевающую «Фиксы после регресса для автотестов» для того, чтобы тесты были пофикшены уже после регресса на следующие спринты, чтобы они не падали

Отчёт составлялся в том случае, если после падения автотестов быстро поднять это не получается по тем или иным причинам (если проблема решается быстро – например, изменились контракты, поменяли урлы, эндпоинты и перезапустили тесты…отчёт не нужен).

Если не все идеально, заказчик уведомляется о существующих проблемах, с целью переноса релиза на некоторое время для фиксации неисправностей

Понятие **Хотфиксы –** обычно используются для того, чтобы подправить какие-то моменты, которые выскочили во время релиза.

**Суть:** когда залили релиз на **Prod** и в нём находятся баги…если бы мы приняли решение исправлять их хотфиксами – его нужно дробить и вырезать какие-то функциональности для исправления. Это чревато тем, что мы откатываем определённые задачи и соответственно нам придётся проводить регресс заново, по той причине, что когда мы вырезаем задачу – мы не можем предугадать, чем это будет чревато и как система поведет себя. Поэтому вырезать задачи и катить (исправлять) их хотфиксами нецелесообразно.

**Решение проблемы: -** либо переносить релиз, либо катить на свой страх и риск. Обычно разработчики с заказчиком смотрят на функциональность, определяют на сколько она критична, озвучиваются возможные риски к чему может привести неработающая функциональность и уже на основе этого принимается решение. В большинстве случае принимается решение о переносе релиза

Далее сборка заливается на стенд **Stage** (**Pre-prod**). Т.к. у нас платежная система, работа с платежами, **Pre-prod** реализован практически таким же образом, как и **Prod** – т.е. это стенд для тестирования каких-либо отдельных фич на сотрудниках (например, происходит тестирование проведения транзакций на реальных картах). Если это уже не первый релиз мы проверяем, что на этом стенде всё работает абсолютно точно так же, как и на **Prod –** основная функциональность не сломалась и всё идёт хорошо (Smoke тестирование). Затем принимается решение, что можно делать релиз, определяется сотрудник, ответственный за это.

Чаще всего релиз проводится по требованию заказчика, глубокой ночью, когда нагрузка на систему не большая, клиенты предупреждаются о возможных неполадках. Релиз проводят несколько **Devops** со стороны заказчика, а также привлекается разработчик со стороны Andersen (иногда люди меняются, ответственные за релиз…своего рода дежурства…для того чтобы на одном человеке не было слишком много ответственности, чтобы в случае чего была взаимозаменяемость)

После релиза происходит проведение смоук тестирования на **Prod** на реальных данных **(**в идеале) – мы его проводили и покрывали автотестами

**Usability testing –** чаще всего ПО разрабатывается по модели Behaviour driven development, т.е. опираясь на сценарии пользователя. В этом случае, к примеру, бизнес-аналитик разрабатывает требования (например, кнопка оплаты должна находится там-то там-то), создаются какие-либо прототипы или макеты. Опираясь на них и ставя себя на место пользователя, мы тестировали удобство интерфейса, личное отношение и мнение по поводу того, удобен ли интерфейс

**О распределении функционала для тестирования между участниками команды** – обычно этим занимаются либо лид, либо так же может быть проявляется личная инициатива, если хочется вдруг протестировать определённый функционал

**Как определить, что работа сделана хорошо** – для подобных ситуации существует такой документ как **Definition of done,** в котором указаны пункты, которые должны быть обязательно выполнены для того чтобы убедиться в том, что мы провели все требующие процедуры, входящие в наши обязанности.

Например, на все задачи из бэклога написаны тест-кейсы, все валидации протестированы позитивно и негативно, функциональное тестирование завершено, регресс прошел успешно, документация обновлена, закрыты все дефекты и т.д.

Также можно использовать такую технику тест-дизайна как **матрица трассируемости,** для того чтобы определить покрыты ли все требования тест-кейсами.