1. 1) Модульное тестирование. На этом уровне проверяется функциональность части приложения. Часто разработчики самостоятельно проводят тестирование написанных модулей с помощью автотестов. В качестве примера можно привести функциональное автоматизированное тестирование формы ввода email при регистрации на сайте.

2) Интеграционное тестирование. Проверяется взаимодействие модулей между собой и их интеграция в общую систему. Тестируют программисты и тестировщики, тестировщик может отправлять тест-кейсы для автоматизации тестирования разработчикам, или же самостоятельно писать код для них/тестировать вручную. Для примера можно представить ручное функциональное тестирование всей формы регистрации на сайте.

3) Системное тестирование – тестирование всей системы в целом, как функциональное, так и не функциональное. Обычно на этом уровне сосредоточена основная работа тестировщика, автоматизировать тестирование становится сложнее. Например, на этом этапе можно произвести нагрузочное тестирование готового сайта.

4) Приемочное тестирование. Проводится исходя из бизнес-требований заказчика и приемочных критериев. Если заказчик или другое уполномоченное лицо не принимает готовый проект, то он отправляется на доработку/более подробное тестирование.

1. Регрессионное тестирование используют для нахождения новых багов в уже протестированных участках кода. Такое может происходить после добавления изменений в программу, соответственно, выполняется оно для нового билда продукта. Обычно для регрессионного тестирования есть хорошо прописанная документация, как документация о функционале программы, так и готовые списки проверок с хорошо прописанными шагами. Регрессионное тестирование хорошо автоматизируется, также опытный тестировщик может проверять не всю программу, а только ее части, в зависимости от внесенных в код изменений, хотя тут всегда могут быть просчеты.
2. Юнит-тесты зачастую пишут разработчики для автоматизированной проверки отдельных модулей кода программы. Преимущество юнит-тестов заключается в возможности отлова багов на самой ранней стадии разработки, тем самым экономя огромное количество времени на более поздних этапах тестирования. Ограничиться одними лишь юнит-тестами нельзя, поскольку продукт зачастую делает распределенная команда, у которой могут быть разные взгляды на продукт/разработку, и при интеграции частей кода в одну программу могут (и должны) появиться баги.
3. Для выбора вида тестирования конкретного продукта нужно, во-первых, знать специфику проекта. Например, для банковских приложений нужно намного более тщательное тестирование, нежели для очередного билда оффлайн-игры, а также и виды тестирования будут отличаться. Для банковского приложения нужно будет провести нагрузочное тестирование, тестирование безопасности, стресс-тестирование. Также нужно знать устройство рабочего процесса в команде, для agile-команд возможно потребуется меньшее количество документации, частое smoke-тестирование и тестирование потенциальных проблемных частей кода на основе опыта тестировщика.
4. Black box testing отличается от white box testing тем, что в первом случае у тестировщика нет доступа к исходному коду, тестировать можно только по входным и выходным данным. При тестировании черного ящика нельзя провести, например, тестирование исходного кода.

Часть 2. Пять примеров видов тестирования сайта <https://spreecommerce-demo.herokuapp.com/>.

1. Функциональное тестирование. Огромное количество тест-кейсов для всего функционала сайта.
2. Тестирование интерфейса исходя из стандартов разработки интернет-магазинов и ожиданий среднестатистического пользователя.
3. Нагрузочное тестирование сайта с помощью виртуальных пользователей.
4. Автоматизированное позитивное тестирование. Проверка базового функционала сайта с точки зрения пользователя, который делает все правильно/не пытается сломать.
5. Ручное негативное тестирование. Тестирование незапланированных действий, например, ввод спец. символов в строку поиска, тестирование невалидной информации на панели регистрации с помощью классов эквивалентности.