***Занятие 2.***

***Чек-лист*** - это список проверок, в которых показано что (какую часть функционала) мы будем тестировать. С указанием результата (passed/failed) и статуса данных проверок.

1. Что мы собрались делать
2. Что мы не хотим забыть
3. Что мы будем проверять (тестировать)

*Чек-лист может представлять собой таблицу, состоящую из:*

1. Number
2. Checking
3. Result (Passed/Failed/Skipped)
4. Comment + ссылка на обнаруженный баг при необходимости

*Для дополнительного описания(полноты) можно использовать следующие данные:*

1. Application
2. Tester
3. Environment
4. Test date
5. Build

Самый простой пример чек-листа это список продуктов для покупки в магазине.

Чек-лист является простым (примитивным) вариантом тест-кейса, который составляется для быстрой проверки основного функционала (без подробного описания шагов и ожидаемых результатов), также может использоваться для регрессионного тестирования.

Чек-лист не стоит использовать, если в команде неопытный сотрудник, который может не понять краткое содержание проверок в виде чек-листа, т.к имеет мало компетенций в бизнес-логике проекта например. то есть чек-листы чаще всего используются опытными сотрудниками, которые хорошо знают проект и могут использовать чек-листы для быстроты тестирования.

***Основные отличия тест-кейса от чек-листа:***

1. Тест-кейс содержит пошаговый сценарий (чек-лист не содержит)
2. Тест-кейс предполагает низкоуровневое (более подробное) тестирование нашего продукта. В то время как чек-лист относится к высокоуровневой (не подробной/поверхностной) документацией.
3. Тест-кейс в большей степени отвечает на вопрос Как? (как мы будем тестировать), что содержит подробное описание того как мы тестируем.
4. Дополнительные атрибуты тест-кейса (которые не используются при составлении чек-листа): Результат наших проверок (ожидаемый результат); Заголовок; Шаги

***Критерии выбора документации Чек-лист или Тест-кейс:***

1. В простом проекте используем Чек-лист. Если же проект содержит большое кол-во требований, которые сложны в реализации и в целом проект является сложным, тогда используется Тест-кейс.
2. Длительность проекта. Проект НЕ длительный = используется Чек-лист. Если же проект длительный = используются тест-кейсы.
3. Размер проекта. Проект, в котором задействовано большое кол-во людей с возможностью прихода новых сотрудников (которых в дальнейшем нужно будет обучать) = используются Тест-кейсы. В проекте с постоянными сотрудниками можно использовать Чек-листы.

При выборе документации желание заказчика является приоритетом. И если специалист считает, что лучше использовать тест-кейсы, но заказчик хочет использовать в качестве документации Чек-листы. Предпочтение отдается чек-листам.

***Виды чек-листов:***

1) **Чек-лист назначения ресурсов и обучения.** Составляется чтобы убедиться, что проект тестирования имеет достаточный бюджет и человеческие ресурсы, выделенные для проекта тестирования. Необходимо проанализировать навыки и компетенции тестовой команды, чтобы убедиться, достаточно ли они компетентны или требуют дополнительной подготовки, чтобы соответствовать требуемому набору навыков.

2) **Чек-лист необходимых инструментов тестирования.** Составляется для выявления нужных программ с соответствующими лицензиями.

3) **Чек-лист по документации тестирования программного обеспечения.** Чек-лист про все функциональные и проектные документы. Они должны быть заполнены, прежде чем команда тестирования сможет приступить к написанию тестовых примеров. В такой чек-лист входит: написание тест-плана, создание тест-кейсов, а также туда можно включить настройки и конфигурации, создание разных матриц, написание метрик.

4) **Чек-лист контрольного списка для тестирования программного обеспечения.** Такой чек-лист может содержать список специалистов и их каждодневные работы. Цель - понять и удостовериться, что все идет по плану.

***Тестирование***

***1.Регрессионное тестирование (Regression)*** проводится после исправления кода, обновлений или любого другого обслуживания системы, чтобы убедиться, что новый код не повлиял на существующий код. После изменения в одной функциональности производится проверка всех функциональностей проекта.

Включает в себя:

* Дымовое Тестирование **(Smoke)**
* Санитарное Тестирование **(Sanity)**

***1.1.Smoke тестирование*** направлено на проверку самой главной, самой важной, самой ключевой функциональности, неработоспособность которой делает бессмысленной саму идею использования программного обеспечения. ***Только позитивные проверки.*** Выполняются каждый раз, когда мы получаем новый билд (версию), проекта (системы) на тестирование, при этом считая её относительно нестабильной. По итогам smoke тестирования программное обеспечение либо отправляется на последующее полное тестирование, либо делается вывод о необходимости его доработки. Оно выполняется до запуска каких-либо подробных функциональных или регрессионных тестов. Основная цель smoke тестирования — отклонить программное приложение с дефектами, чтобы команда QA не тратила время на тестирование программного приложения с неработающим основным функционалом. Цель состоит в том, чтобы убедиться, что критические функциональные возможности системы работают нормально.

***1.2.Sanity*** - тестирование, выполняемое после получения более стабильной сборки ПО с незначительными изменениями в коде или функциональности, чтобы убедиться, что ошибки были исправлены и никаких дальнейших проблем не возникает из-за этих изменений. Используется каждый раз, когда мы получаем относительно стабильную сборку ПО, чтобы определить работоспособность в деталях. Цель состоит в том, чтобы определить, что предлагаемая функциональность работает в целом так, как ожидалось. Если sanity тест не проходит, сборка отклоняется, чтобы сэкономить время и затраты, связанные с более тщательным тестированием.

**Ключевые отличия Sanity и Smoke тестирования**

| **Smoke** | **Sanity** |
| --- | --- |
| * Исполняется чтобы убедиться, что критически важные функциональные возможности программы работают, как положено * Цель – проверить «стабильность» системы в целом, чтобы приступить к более тщательному тестированию * Проверяет всю ключевую функциональность в системе * Похоже на общую проверку * Выполняется перед регрессионным тестированием * Может выполняться автоматизированно или вручную | * Нацелено на проверку того, что определённые части всё так же работают, как и положено, после минорных изменений или исправлений багов * Цель – проверить общее состояние системы в деталях, чтобы приступить к более тщательному тестированию * Проверяет только функциональность конкретного компонента системы * Похоже на специализированную, углубленную проверку * Выполняется перед регрессионным тестированием и после проведения smoke тестов * Чаще выполняется вручную |

***Также необходимо запомнить следующие моменты:***

1. И Sanity и Smoke тестирование - это способы избежать потери времени и усилий, помогают быстро определить недостатки ПО и их критичность, а также то, заслуживает ли оно перехода в фазу более углубленного и тщательного тестирования или же нет.
2. Sanity и Smoke тестирование не нацелены на перепроверку дефектов.
3. Smoke тестирование, выполняемое на конкретной сборке.
4. Одной из лучших отраслевых практик является ежедневное Sanity и Smoke тестирование.
5. Sanity и Smoke тестирование может выполняться вручную или с помощью инструментов автоматизации. Когда используются автоматические инструменты, тесты часто инициируются тем же процессом, который генерирует саму сборку.
6. В соответствии с потребностями тестирования вам, возможно, придется выполнить как Sanity, так и Smoke тесты в сборке программного обеспечения. В таких случаях вы сначала выполняете Smoke тесты, а затем приступаете к Sanity тестированию.
7. На практике тест-кейсы для Sanity тестирования обычно объединяются со Smoke тестами, для того чтобы ускорить выполнение тестирования. Следовательно, часто эти термины путаются и используются взаимозаменяемо.

| **Дымовые (Smoke)** | **Санити (Sanity)** | **Регрессионные (Regression)** |
| --- | --- | --- |
| Исполняются с целью проверить что критически важные функциональные части AUT работают как положено | Нацелено на установление факта того, что определённые части AUT всё так же работают как положено после минорных изменений или исправлений багов | Подтверждают, что свежие изменения в коде или приложении в целом не оказали негативного влияния на уже существующую функциональность/набор функций |
| Цель — проверить «стабильность» системы в целом, чтобы дать зелёный свет проведению более тщательного тестирования | Целью является проверить общее состояние системы в деталях, чтобы приступить к более тщательному тестированию | Цель — убедиться что свежие изменения в коде не оказали побочных эффектов на устоявшуюся работающую функциональность |
| Перепроверка дефектов не является целью Smoke | Перепроверка дефектов не является целью Sanity | Перепроверка дефектов не является целью Regression |
| Дымовое тестирование выполняется перед регрессионным | Санитарное тестирование выполняется перед регрессионным и после smoke-тестов | Проводится на основании требований проекта и доступности ресурсов (закрывается автотестами), «регресс» может проводиться в параллели с ре-тестами |
| Может выполняться автоматизированно или вручную | Чаще выполняется вручную | Лучший повод для автоматизации данного вида тестирования, т.к. ручное может быть крайне затратным по ресурсам или времени |
| Является подмножеством регрессионного тестирования | Подмножество приемочного тестирования | Выполняется при любой модификации или изменениях в уже существующем проекте |
| Тест-кейсы часть регрессионных тест-кейсов, но покрывающие крайне критичную функциональность | Санитарное может выполняться без тест-кейсов, но знание тестируемой системы обязательно | Тест-кейсы регрессионного тестирования могут быть получены из функциональных требований или спецификаций, пользовательских мануалов, и проводятся вне зависимости от того, что исправили разработчики |

***2. Критического пути (critical path test). Подраздел End-to-End тестирвания*** — основной тип тестовых испытаний, во время которого значимые элементы и функции приложения проверяются на предмет правильности работы при стандартном их использовании (типичными пользователями в типичной повседневной деятельности).

Чаще всего на практике на данном уровне тестирования проверяется основная масса требований к продукту. Пример: возможность набора текста, вставки картинок, возможность войти на сайт, создать запись, и т.д.

Для данного вида тестирования пишутся наиболее подробные и глубокие тест-кейсы, чтобы покрыть всю возможную функциональность приложения. Тест критического пути может быть как позитивным, так и негативным.

Тест критического пути является одним из самых распространенных видов функционального тестирования. Частота проведения данного тестирования обусловлена в первую очередь необходимостью периодической проверки всего приложения в сжатые сроки. А также позволяет выявить самые быстро находимые дефекты и исправить приложение в более сжатые сроки.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Как правило DDOS атакам подвергаются приложения/сервисы, у которых есть статичный IP-адрес. Если у приложения динамический IP-адрес (мобильное приложение), то оно не подойдет под DDOS атаку. Статичный IP обычно есть у серверных приложений/бэкенда.

Мобильное приложение обычно DDOS-атакам не подвергается. DDOS-атака представляет собой большое количество запросов, которые настроены на 1 IP-адрес (т.е. статичный).

Даже если дома с Wi-fi сижу в мобильном приложении - все равно будет динамический IP.

End to end тест смысл = когда заканчиваю пользоваться приложением (сворачиваешь его)

CPT - развернутый смоук (проверить кликабельность кнопки/сворачивание и раскрытие приложения)







