1. Тестирование. Качество ПО

Тестирование:

- проверка соответствия между реальным и ожидаемым поведением.

Так же Тестирование :

- это одна из техник контроля качества, включающая в себя активности по:
- Test Management (планированию работ).
- Test Design (проектирование тестов).
- Test Execution (выполнение тестов).
- Test Analysis (анализ результатов тестирования).

Качество ПО:

- это совокупность характеристик программного обеспечения, относящихся к его способности удовлетворять установленные и предполагаемые потребности.

2. Валидация vs Верификация

Верификация (verification):

- оценка соответствия продукта требованиям (спецификации).

Валидация (validation):

- оценка соответствия продукта ожиданиям и требованиям пользователей.

3. Цели тестирования:

- Повысить вероятность того, что приложение:
- будет соответствовать всем описанным требованиям.
- будет работать правильно при любых обстоятельствах.
- Предоставление актуальной информации о состоянии продукта на данный момент.

4. Этапы тестирования:

1. Анализ продукта
2. Работа с требованиями
3. Разработка тест плана
4. Создание тестовой документации
5. Тестирование
6. Отчет о тестировании

7. Стабилизация
8. Эксплуатация

5. Тест план

Test Plan - это документ, описывающий весь объем работ по тестированию

Отвечает на вопросы:

- Что?
- Когда?
- Критерии начала/окончания тестирования.
- Окружение (environment) dev/staging/production?
- Подходы/техники/инструменты/виды тестирования?
- Браузеры/версии/OS/разрешения экрана?
- Кто? Обязанности? Ресурсы? Обучение?
- Сроки?
- График?
- Стратегия тестирования.
- Ссылки на документацию.
- Ссылки на требования.

6. Тест дизайн

Test design — это этап процесса тестирования ПО, на котором проектируются и создаются тест кейсы, в соответствии с критериями качества и целями тестирования.

7. Техники тест дизайна

Эквивалентное Разделение (Equivalence Partitioning)

- Как пример, у вас есть диапазон допустимых значений от 1.00 до 10.00 долларов, вы должны выбрать одно любое верное значение внутри интервала, скажем, 5.00, и любые неверные значения вне интервала, например 0.99 и 11.00.

Анализ Граничных Значений (Boundary Value Analysis)

- Как пример, у вас есть диапазон допустимых значений от 1.00 до 10.00 долларов.
- Two value (двузначный) BVA: валидные граничные значения 1.00, 10.00, и невалидные значения 0.99 и 10.01.

- Three/Full value (трехзначный) BVA: валидные граничные значения 1.00, 1.01, 10.00, 9.99, и невалидные значения 0.99 и 10.01.

Причина / Следствие (Cause/Effect)

- ввод комбинаций условий (причин), для получения ответа от системы (следствие).
- Например, вы проверяете возможность добавлять клиента:
- **Причина:** необходимо заполнить поля «Имя», «Адрес», «Номер Телефона» и нажать кнопку «Добавить».
- **Следствие:** После нажатия на кнопку «Добавить», система добавляет клиента в базу данных и показывает его номер на экране.

Предугадывание ошибки (Error Guessing)

- использование знаний системы и способность к интерпретации спецификации на предмет того, чтобы «предугадать» при каких входных условиях система может выдать ошибку.
- Например, спецификация говорит: «пользователь должен ввести код».
- Тестировщик будет думать: «Что, если я не введу код?», «Что, если я введу неправильный код?»…

8. Продвинутые техники тест дизайна

Попарное тестирование (Pairwise Testing)

- Формирование таких наборов тестовых данных, в которых каждое тестируемое значение каждого из проверяемых параметров хотя бы единожды сочетается с каждым тестируемым значением всех остальных проверяемых параметров.
- Суть техники — мы не проверяем все сочетания всех значений, но проверяем все пары значений.

Таблица принятия решений (Decision table)

- В таблицах решений представлен набор условий, одновременное выполнение которых должно привести к определенному действию/решению.

Диаграмма (граф) состояний-переходов (State Transition diagram)

- Диаграмма для описания поведения системы.
- Система имеет конечное число состояний и переходов между ними.
- Диаграмма может быть переведена в Таблицу состояний-переходов (или в таблицу принятия решений).

Use case (пользовательский сценарий)

- Это сценарий взаимодействия пользователя с системой для достижения цели.

Use case содержит:

- кто использует систему (например роль админ/покупатель/продавец).
- что пользователь хочет сделать.
- цели пользователя.
- шаги, которые выполняет пользователь.
- описание того, как система реагируют на действия пользователя.

9. Exploratory vs Ad-hoc testing

Исследовательское тестирование (exploratory testing)

- это одновременное изучение системы, проектирование тестов (тест дизайн) и тестирование.
- Данная техника базируется на опыте тестировщика (experience based).
- Пример: приходит тестировщик на новый проект и начинает одновременно изучать сайт, писать чек-лист и проходить этот чек-лист (тестировать).

Ad-hoc тестирование

- Перевод от автора статьи - “тестирование от балды”.
- Вид тестирования, который выполняется без подготовки к тестам, без определения ожидаемых результатов, проектирования тестовых сценариев.
- Неформальное, импровизационное тестирование.

10. Test Case (тестовый случай)

Test Case: это артефакт/документ, описывающий совокупность шагов, конкретных условий и параметров, необходимых для проверки тестируемой функции.

Тест кейс состоит из:

- ID (идентификатор)
- Title (название)
- Type (тип)
- Priority (приоритет)
- Preconditions (предусловия)
- Steps (шаги)
- Expected Result (ожидаемый результат)
- Post conditions (пост условия) - например очистка данных или возвращение системы в первоначальное состояние.

Тест кейсы разделяются на позитивные и негативные:

- **Позитивный тест кейс** использует только корректные данные и проверяет, что приложение правильно выполнило вызываемую функцию.

- **Негативный тест кейс** оперирует как корректными, так и некорректными данными (минимум 1 некорректный параметр) и ставит целью проверку исключительных ситуаций (срабатывание валидаторов), а также проверяет, что вызываемая приложением функция не выполняется при срабатывании валидатора.

11. Check-list (Чек-лист)

Check list

- это документ, описывающий что должно быть протестировано.

- Чек-лист может быть абсолютно разного уровня детализации.
- Как правило, чек-лист содержит только действия (шаги) без ожидаемого результата.
- Чек-лист менее формализован чем тест кейс.

12.Bug report (баг репорт)

Bug Report

- это документ, описывающий последовательность действий, которые привели к некорректной работе системы, с указанием причин и ожидаемого результата.

Основные составляющие Bug report:

- ID (идентификатор)
- Название (Title)
- Короткое описание (Summary)
- Проект (Project)
- Компонент приложения (Component)
- Номер версии (Version)
- Серьезность (Severity)
- Приоритет (Priority)
- Статус (Status)
- Автор (Author)
- Назначен на (Assignee)
- Окружение (Environment)
- App/build version (версия билда/приложения)
- Шаги воспроизведения (Steps to Reproduce)
- Фактический Результат (Actual Result)
- Ожидаемый результат (Expected Result)

Дополнительные составляющие Bug report:

- Screenshots (скриншоты)
- Video (видео)
- Credentials (login + password)
- Browser console errors (логи с браузера)
- Mobile app logs (логи с мобилки)
- Server logs (логи с сервера)
- Requests (запросы)
- Analytics events (ивенты с аналитики)
- Database data (данные из базы данных)
- Database queries (запросы в базу)
- Date and time (дата и время)
- Comments/Notes (комментарии/заметки)
- Link tasks/bugs (подвязка других задач/багов к текущему)

13.Severity vs Priority

Серьезность (Severity)

- это атрибут, характеризующий влияние дефекта на работоспособность приложения. Severity выставляется тестировщиком.

Градация Severity:

- S1 Блокирующая (Blocker)
- S2 Критическая (Critical)

- S3 Значительная (Major)
- S4 Незначительная (Minor)
- S5 Тривиальная (Trivial)

Приоритет (Priority)

- это атрибут, указывающий на очередность выполнения задачи или устранения дефекта. Чем выше приоритет, тем быстрее нужно исправить дефект.

Градация Priority:

- P1 Высокий (High)
- P2 Средний (Medium)
- P3 Низкий (Low)

14.Traceability matrix (Матрица соответствия требований)

- Traceability matrix - это двумерная таблица, содержащая соответствие функциональных требований и тест кейсов.
- В заголовках колонок таблицы расположены требования, а в заголовках строк — ID тест кейсов.
- На пересечении — отметка, означающая, что требование текущей колонки покрыто тестовым сценарием текущей строки.

15.Defect / Error / Bug / Failure

Дефект (он же баг)

- это несоответствие фактического результата ожидаемому результату, описанному в требованиях.

Bug (defect)

- ошибка программиста (или другого члена команды), то есть когда в программе, что-то идёт не так как планировалось и программа выходит из-под контроля.

Error

- ошибка пользователя, то есть он пытается использовать программу иным способом.

Failure

- сбой (причём необязательно аппаратный) в работе компонента, всей программы или системы.

16.Уровни Тестирования (Levels of testing)

1. **Модульное тестирование (Unit Testing)**
   - Тестирование объектов, классов, функций и т.д., обычно выполняемое программистами.

2. **Интеграционное тестирование (Integration Testing)**
   - Тестирование взаимодействия между классами, функциями, модулями. Например тестирование API через Postman.

3. **Системное тестирование (System Testing)**
   - Проверка как функциональных, так и не функциональных требований в системе.

4. **Приемочное тестирование (Acceptance Testing)**
   - Проверка соответствия системы требованиям и проводится с целью:
   - определения удовлетворяет ли система приемочным критериям;
   - вынесения решения заказчиком/менеджером принимается приложение или нет.

17.Виды / типы тестирования (Testing types)

Функциональные виды тестирования:

- Функциональное тестирование (Functional testing)
- Тестирование пользовательского интерфейса (GUI Testing)
- Тестирование безопасности (Security and Access Control Testing)
- Тестирование взаимодействия (Interoperability Testing)

Нефункциональные виды тестирования:

- Все виды тестирования производительности (Performance):
- нагрузочное тестирование (**Load Testing**) - много пользователей.
- стрессовое тестирование (**Stress Testing**) - очень много данных и/или пользователей (пиковые значения).
- объемное тестирование (**Volume Testing**) - много данных.

- тестирование стабильности или надежности (**Stability / Reliability Testing**)
- Тестирование установки (**Installation testing**)
- Тестирование удобства пользования (**Usability Testing**)
- Тестирование на отказ и восстановление (**Failover and Recovery Testing**)
- Конфигурационное тестирование (**Configuration Testing**)

Связанные с изменениями виды тестирования:

- Дымовое тестирование (**Smoke Testing**)
- Регрессионное тестирование (**Regression Testing**)
- Повторное тестирование (**Re-testing**)
- Тестирование сборки (**Build Verification Test**)
- Санитарное тестирование или проверка согласованности/исправности (**Sanity Testing**)

18.Принципы тестирования (Principles of testing)

1. **Тестирование демонстрирует наличие дефектов (Testing shows presence of defects)**
    - Тестирование может показать, что дефекты присутствуют, но не может доказать, что их нет.

2. **Исчерпывающее тестирование недостижимо (Exhaustive testing is impossible)**
    - Полное тестирование с использованием всех комбинаций вводов и предусловий физически невыполнимо, за исключением тривиальных случаев.

3. **Раннее тестирование (Early testing)**
    - Чтобы найти дефекты как можно раньше, активности по тестированию должны быть начаты как можно раньше в жизненном цикле разработки.

4. **Скопление дефектов (Defects clustering)**
    - Как правило, большая часть дефектов, обнаруженных при тестировании, содержится в небольшом количестве модулей.

5. **Парадокс пестицида (Pesticide paradox)**

- Если одни и те же тесты будут прогоняться много раз, в конечном счете этот набор тестовых сценариев больше не будет находить новых дефектов.

6. **Тестирование зависит от контекста (Testing is context dependent)**
    - Тестирование выполняется по-разному в зависимости от контекста.

7. **Заблуждение об отсутствии ошибок (Absence-of-errors fallacy)**
    - Обнаружение и исправление дефектов не помогут, если созданная система не подходит пользователю и не удовлетворяет его ожиданиям и потребностям.

19.Статическое и динамическое тестирование

Статическое (static) тестирование

- Производится **БЕЗ** запуска кода продукта.
- Примеры: тестирование требований/документации, код ревью, статические анализаторы кода.

Динамическое (dynamic) тестирование

- Производится С запуском кода продукта.

20.Требования (requirements)

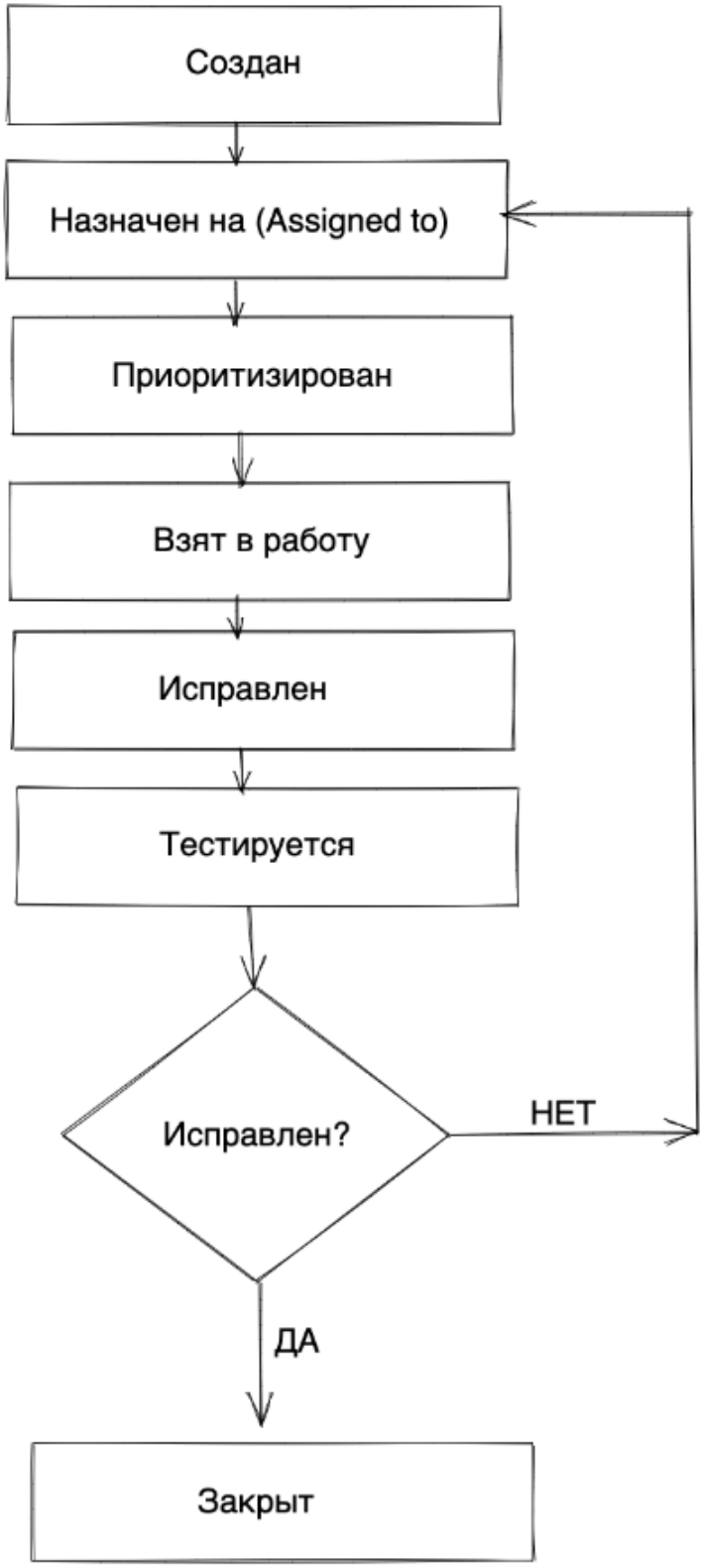
Требования - это спецификация (описание) того, что должно быть реализовано.
Требования описывают то, что необходимо реализовать, без детализации технической стороны решения. Что, а не как.

Требования к требованиям:

1. корректность
2. недвусмысленность
3. полнота
4. непротиворечивость
5. упорядоченность по важности и стабильности
6. проверяемость (тестопригодность)
7. модифицируемость
8. трассируемость

9. понимаемость

21.Жизненный цикл бага



22.Жизненный цикл разработки ПО

Software Development Life Cycle (SDLC):

1. Идея (Idea)
2. Сбор и анализ требований (Planning and Requirement Analysis)
3. Документирование требований (Defining Requirements)
4. Дизайн (Design Architecture)
5. Разработка (Developing)
6. Тестирование (Testing)
7. Внедрение/развертывание (Deployment)
8. Поддержка (Maintenance)
9. Смерть (Death)