1. История тестирования программного обеспечения

2. Обеспечение качества: терминология, ISO/IEC 25010:2011 (Модель качества прииспользовании)

3. Модель качества продукта по стандарту ISO/IEC 25010:2011 (Ответ 2 вопроса!)

4. Процесс и методологии разработки программного обеспечения (Водопадная, V- образная, Итерационная)

5. Процесс и методологии разработки программного обеспечения (Спиральная, Гибкая, SCRUM)

6. Процесс и методологии разработки программного обеспечения (Канбан, Экстремальное программирование)

7. Процесс тестирования программного обеспечения (Планирование и управление тестированием, Анализ и проектирование тестов, Внедрение и реализация тестов, Оценка критериев выхода и создание отчетов)

8. Уровни тестирования программного обеспечения, Верификация и валидация

9. Типы тестирования программного обеспечения (Статическое и динамическое, Ручное и автоматизированное)

10. Методы тестирования программного обеспечения (Белого, черного, серого ящиков)

11. Виды тестирования программного обеспечения (функциональное, нефункциональное, структурное, тестирование изменений, тестирование по приоритетам)

12. Функциональное тестирование, Модели поведения пользователя, Позитивное и негативное тестирование

13. Исследовательское тестирование ПО (что это, отличие от сценарного тестирования, когда стоит и когда не стоит применять)

14. Исследовательское тестирование ПО (Туры, Туры по бизнес-центру)

15. Исследовательское тестирование ПО (Туры, Туры историческому району, Туры по району развлечений)

16. Исследовательское тестирование ПО (Туры, Туры по туристическому району, Туры по району отелей)

17. Исследовательское тестирование ПО (Туры, Туры по неблагополучному району)

18. Тестирование требований (Почему важно? Что тестируется? Параметры тестирования документации)

19. Тестирование требований (Уровни и виды требований, Критерии качества требований, Методы тестирования требований)

20. Нефункциональное тестирование (Что это, Его параметры, и виды. Инсталляционное тестирование)

21. Нефункциональное тестирование (Тестирование производительности)

22. Нефункциональное тестирование (Тестирование эргономичности, Тестирование совместимости, Тестирование GUI)

23. Нефункциональное тестирование (Тестирование глобализации, A/B

тестирование, Тестирование на отказ и восстановление системы, Тестирование на соответствие стандартам)

24. Тестирование безопасности (Что это, Причины пробелов в безопасности, Тестирование проницаемости)

25. Тестирование безопасности (Виды атак)

26. Тестирование безопасности (Виды вредоносного ПО)

27. Тестирование безопасности (Тестирование на проникновение)

28. Тестовая документация (Ее виды и назначение, Тест план)

29. Тестовая документация (График тестирования, Матрица трассируемости, Тестовый набор, Чек-лист)

30. Техники тест дизайна (Что это, Методы черного ящика: классы эквивалентности, Анализ граничных значений)

31. Техники тест дизайна (Методы черного ящика: таблица решений, таблица (диаграмма) переходов)

32. Техники тест дизайна (Методы черного ящика: тестирование по вариантам использования, попарное тестирование)

33. Техники тест дизайна (Методы белого ящика: тестирование покрытия операторов, тестирование покрытия ветвей. Методы, основанные на опыте)

34. Тестовые случаи (Цели, Жизненный цикл, Атрибуты тестовых случаев)

35. Тестовые случаи (Свойства качественных тестовых случаев, типичные ошибки при создании тестовых случаев)

36. Тестовые сценарии (Цели, Шаги для создания, Атрибуты тестовых сценариев)

37. Тест план, его назначение и структура

38. Отчеты об ошибках (Цель, Жизненный цикл, Атрибуты)

39. Отчеты об ошибках (Свойства качественных отчетов об ошибках, Типичные ошибки при написании)

40. Отчеты о результатах тестирования

41. Автоматизация тестирования (Виды, Модульное тестирование)

42. Автоматизация тестирования (Интеграционные тесты, UI тестирование)

1. История тестирования программного обеспечения

Первые программные системы разрабатывались в рамках программ научных исследований или программ для нужд министерств обороны.

В 1960-х много внимания уделялось «исчерпывающему» тестированию, которое должно проводиться с использованием всех путей в коде или всех возможных входных данных.

Однако это невозможно:

#1 количество возможных входных данных очень велико;

#2 существует множество путей;

#3 сложно найти проблемы в архитектуре и спецификациях.

В начале 1970-х годов тестирование программного обеспечения обозначалось как «процесс, направленный на демонстрацию корректности продукта» или как «деятельность по подтверждению правильности работы программного обеспечения».

Во второй половине 1970-х тестирование представлялось как выполнение программы с намерением найти ошибки, а не доказать, что она работает.

В 1980-е годы тестирование расширилось таким понятием, как предупреждение дефектов.

Стали высказываться мысли, что необходима методология тестирования, в частности, что тестирование должно включать проверки на всем протяжении цикла разработки, и это должен быть управляемый процесс.

В ходе тестирования надо проверить не только собранную программу, но и требования, код, архитектуру, сами тесты.

В начале 1990-х годов в понятие «тестирование» стали включать планирование, проектирование, создание, поддержку и выполнение тестов и тестовых окружений, и это означало переход от тестирования к обеспечению качества.

2. Обеспечение качества: терминология, ISO/IEC 25010:2011 (Модель качества прииспользовании)

Обеспечение качества (Quality Assurance - QA) - это

совокупность мероприятий, охватывающих все технологические этапы разработки, выпуска и эксплуатации программного обеспечения (ПО) информационных систем, предпринимаемых на разных стадиях жизненного цикла ПО, для обеспечения требуемого уровня качества выпускаемого продукта

Стандарт ISO/IEC 25010:2011 (ГОСТ Р ИСО/МЭК 25010-2015)[8] определяет модель качества продукта, которая включает восемь характеристик верхнего уровня:

- **функциональная пригодность** - Способность решать нужный набор задач
- **уровень производительности** - определение масштабируемости приложения под нагрузкой
- **совместимость** - может ли программное обеспечение взаимодействовать с другими программными компонентами, программным обеспечением или системами.
- **удобство использования (юзабилити)** - способность продукта быть понимаемым, изучаемым, используемым и привлекательным для пользователя в заданных условиях
- **надёжность** - свойство объекта сохранять во времени в установленных пределах значения всех параметров, характеризующих способность выполнять требуемые функции в заданных условиях применения
- **защищённость** - защищает ли информационная система данные и поддерживает ли функциональность, как предполагалось
- **сопровождаемость** - процесс улучшения, оптимизации и устранения дефектов программного обеспечения (ПО) после передачи в эксплуатацию
- **переносимость (мобильность)** - способность программного обеспечения работать с несколькими аппаратными платформами или операционными системами.

Модель качества при использовании

- Эффективность (результативность)
- Производительность
- Удовлетворенность (полноценность, доверие, удовольствие, комфорт)
- Свобода от риска (смягчение отрицательных последствий:
  - экономического риска;
  - риска для здоровья и безопасности;
  - экологического риска.
- Покрытие контекста (полнота контекста, гибкость)

3. Модель качества продукта по стандарту ISO/IEC 25010:2011

Качество в использовании - степень применимости продукта или системы заданными пользователями для удовлетворения их потребностей в достижении заданных целей с результативностью, эффективностью, свободой от риска и удовлетворенностью в заданных контекстах использования.

Качество в использовании характеризует влияние, которое продукт оказывает на правообладателей. Оно определяется качеством программного обеспечения, аппаратных средств и эксплуатационной среды, а также характеристиками пользователей, задач и социального окружения

Данная модель состоит из 5 характеристик:

- Результативность - точность и полнота, с которой пользователи достигают заданных целей.
- Эффективность - ресурсы, затрачиваемые в зависимости от точности и полноты, с которыми пользователь достигает целей.
- Удовлетворенность - степень удовлетворения потребностей пользователя при применении продукта или системы в заданном контексте использования. Подхарактеристики: применимость, доверие, удовольствие, комфорт.
- Свобода от риска - степень уменьшения продуктом или системой потенциального риска для экономического статуса, человеческой жизни, здоровья или окружающей среды. Подхарактеристики: уменьшение экономического риска, уменьшение риска для здоровья и безопасности, уменьшение риска для окружающей среды.
- Покрытие контекста - степень применимости продукта или системы с результативностью, эффективностью, свободой от риска и удовлетворенностью как в заданных контекстах использования, так и вне них. Подхарактеристики: покрытия контекста являются полнота контекста, гибкость.

4. Процесс и методологии разработки программного обеспечения (Водопадная, V- образная, Итерационная)

Водопадная модель (waterfall model) в современных проектах практически неприменима. Она предполагает однократное выполнение каждой из фаз проекта, которые, в свою очередь, строго следуют друг за другом.

Цикл разработки ПО:

1. Проектирование -> 2. Дизайн -> 3. Кодирование -> 4. Тестирование -> 5. Поддержка

Благодаря её жесткости, разработка проходит быстро, стоимость и срок заранее определены. Но данная модель будет давать отличный результат только в проектах с четко и заранее определенными требованиями и способами их реализации. Нет возможности сделать шаг назад, тестирование начинается только после того, как разработка завершена или почти завершена. Продукты, разработанные по данной модели без обоснованного ее выбора, могут иметь недочеты (список требований нельзя скорректировать в любой момент), о которых становится известно лишь в конце из-за строгой последовательности действий. Стоимость внесения изменений высока, так как для ее инициализации приходится ждать завершения всего проекта. Тем не менее, фиксированная стоимость часто перевешивает минусы подхода. Исправление осознанных в процессе создания недостатков возможно.

V-образная модель

V-образная модель состоит из Детализации проекта (определение концепции, выбор архитектуры и проектирование основных идей), реализации проекта и тестирование и компоновка проекта (Выполнение модульного тестирования и интеграции фич, тестирование и проверка системы, введение в эксплуатацию и поддержка проекта).

Концепция ---------- введение в эксплуатацию и поддержка
Проектир архитектуры ---------- Тестир и проверка с-мы
Детальое проектирование ---------- Модульное тестир и интегр
Реализация

Итерационная инкрементальная модель

В инкрементной модели полные требования к системе делятся на различные сборки. Терминология часто используется для описания поэтапной сборки ПО. Имеют место несколько циклов разработки, и вместе они составляют жизненный цикл «мульти-водопад». Цикл разделен на более мелкие легко создаваемые модули. Каждый модуль проходит через фазы определения требований, проектирования, кодирования, внедрения и тестирования. Процедура разработки по инкрементной модели предполагает выпуск на первом большом этапе продукта в базовой функциональности, а затем уже последовательное добавление новых функций, так называемых «инкрементов». Процесс продолжается до тех пор, пока не будет создана полная система.

5. Процесс и методологии разработки программного обеспечения (Спиральная, Гибкая, SCRUM)

Спиральная модель (spiral model) представляет собой частный случай итерационной инкрементальной модели, в котором особое внимание уделяется управлению рисками, в особенности влияющими на организацию процесса разработки проекта и контрольные точки.

Спиральная модель предполагает 4 этапа для каждого витка:

- Планирование
- Анализ рисков
- Конструирование
- оценка результата и при удовлетворительном качестве переход к новому витку

Гибкая методологи разработки ПО (Agile):

Ценности:

Люди и взаимодействие людей важнее чем процессы и инструменты (Если какой-то процесс или инструмент облегчает работу людей, то он полезен, иначе нет)

Рабочий софт важнее документации

Взаимодействие с клиентами важнее контракта

Реакция на изменения важнее чем следование плану

Принципы:

Наибольший приоритет на написание раннего рабочего софта(побыстрее выпустить рабочий вариант, чтобы потом можно было его довести до совершенства)

Быть готовым к изменениям

Частое оповещение о прогрессе в разработке продукта заказчику

Заказчики и разработчики должны работать вместе (Продукт делается для них, так что их замечания могут быть ценны)

Стоить проект вокруг инициатив

Лучший способ коммуникации это оффлайн “С глазу на глаз”

Рабочий софт – мерило прогресса.

Разработка должна быть устойчивой. Скорость должна быть быстрой, но стабильной.

Уделяйте внимание хорошим техническим практикам

Простота залог успеха

Лучшая архитектура разработки – самоорганизация

Команда должна показывать регулярную эффективность до внемения правок.

Методология разработки ПО SCRUM:

Изначально имеется список требований (фичи, которые нужно реализовать).

Scrum базируется на спринтах – коротких промежутках времени, за которые реализуют некоторую часть фичи.

Всё, что выполнено за спринт заносится в sprint backlog – вконце нескольких спринтов имеем готовый продукт.

Плюсом этой концепции является, возможность добавить требования и получить результат в течении небольшого времени (1 или нескольког спринтов)

6. Процесс и методологии разработки программного обеспечения (Канбан, Экстремальное программирование)

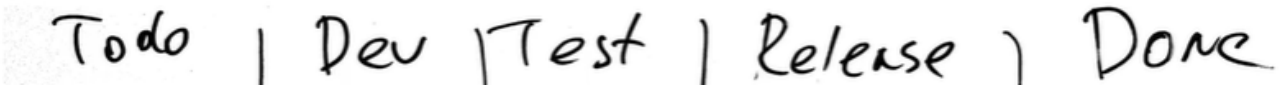
Процесс разработки ПО – структура, согласно которой построено создание нового ПО.

Модели разработки ПО – структура, систематизирующая различные виды проектной деятельности, их взаимодействие и последовательность в процессе разработки ПО

Методология разработки ПО Kanban:

Канбан-доска позволяет вывести процесс выполнения задач в визуальное восприятие. Такой подход помогает видеть весь рабочий процесс, четко распределять задачи и вовремя направлять усилия в «слабые» зоны.

Это работает так: столбики представляют собой разные этапы, на которые разбивают рабочий процесс. Карточки в столбцах — это конкретные задачи-шаги. За каждый этап несет ответственность отдел/сотрудник. Карточки перемещаются по столбцам в соответствии со своим статусом.



Почему Kanban?

Имеется визуализация прогресса, очень легка начать, не нужно каждые 2 недели проводить совещания, всё добавляется и удаляется сразу же.

Методология разработки ПО Extreme Programing:

Цель методики XP — справиться с постоянно меняющимися требованиями к программному продукту и повысить качество разработки. Поэтому XP хорошо подходит для сложных и неопределенных проектов.

Методология XP строится вокруг четырех процессов: кодирования, тестирования, дизайна и слушания. Кроме того, экстремальное программирование имеет ценности: простоту, коммуникацию, обратную связь, смелость и уважение.

Основные тезисы экстремального программирования:

- Планирование
- Короткий релиз
- Метафора – сложные вещи, нужно объяснять простыми примерами
- Простой дизайн
- Тестирование
- Рефакторинг – оптимизация кода
- Парное программирование – Code Review, второй программист проверяет код
- Коллективная собственность – все равны в своих правах на проект
- Непрерывная интеграция
- 40 часовая рабочая неделя
- Клиент или представитель должен быть рядом с программистами, чтобы у него можно было что-то уточнит
- Нужно использовать стандарты кодирования

7. Уровни тестирования программного обеспечения, Верификация и валидация

Уровни тестирования

Компонентное тестирование выполняется во время написания кода. его выполняет сам программист. Следовательно, ошибки, в большинстве случаев, исправляются сразу же и не попадают к специалистам по тестированию.

Данное тестирование направлено на тестирование отдельных модулей и компонентов программы, которые изолированы от других модулей и компонентов.

Для этого уровня тестирования характерно несколько целей: 1. Проверка компонента на соответствие требованиям; 2. Обнаружение ошибок в компоненте; 3. Предотвращение пропуска ошибок на более высокие уровни тестирования.

Интеграционное тестирование проверяет интерфейсы между компонентами, взаимодействие различных частей системы, таких как операционная системы, файловая система, аппаратное обеспечение, и интерфейсы между системами. Интеграционное тестирование может состоять из одного или более уровней и может быть выполнено на тестовых объектах разного размера следующим образом:

1. Компонентное интеграционное тестир проверяет взаимодействие между программными компонентами и производится после компонентного тестирования.

2. Системное интеграционное тестирование проверяет взаимодействие между системами или между аппаратным обеспечением и может быть выполнено после системного тестирования. В этом случае, разработчики могут управлять только одной стороной интерфейса.

Системное тестирование При системном тестировании наша задача уже состоит в том, чтобы убедиться в корректности работы в целом всей системы. Программа в этом случае должна быть максимально приближена к конечному результату. А наше внимание должно быть сосредоточено на общем поведении системы с точки зрения конечных пользователей. Для этого уровня тестирования также характерно несколькоцелей: 1. Проверка системы на соответствие требованиям; 2. Обнаружение ошибок в системе; 3. Предотвращение пропуска ошибок на более высокие уровни тестирования.

Приемочным тестированием Этот уровень тестирования используется для подтверждения готовности продукта и проводится преимущественно в самом конце цикла разработки программы. У приемочного тестирования есть также несколько целей: 1. Показать, что программа завершена и готова к использованию так, как от нее ожидалось; 2. Проверить, что работа программы соответствует установленному ТЗ или требованиям.

Целью **верификации** является достижение гарантии того, что верифицируемый объект (требования или программный код) соответствует требованиям, реализован без непредусмотренных функций и удовлетворяет проектным спецификациям и стандартам.(компонентное, интеграционное, системное тестирования)

Валидация - процесс, целью которого является доказательство того, что в результате разработки системы мы достигли тех целей, которые планировали достичь благодаря ее использованию (это проверка соответствия системы ожиданиям заказчика). (приемочное тестирование)

8. Процесс тестирования программного обеспечения (Планирование и управление тестированием, Анализ и проектирование тестов, Внедрение и реализация тестов, Оценка критериев выхода и создание отчетов)

Планирование тестирования - это действия, направленные на определение целей тестирования и описание задач тестирования для достижения этих целей и миссии.

Управление тестированием - это постоянное сопоставление текущего положения дел с планом и отчетность о состоянии дел, включая отклонения от плана.

Планирование тестирования ПО:

- Анализ требований;
- Определение целей тестирования;
- Определение общего подхода к тестированию (уровни тестирования, виды тестирования, критерия входа в тестирование);
- Интегрирование с разработкой ПО (требования, архитектура, дизайн, разработка, тестирование, релиз);
- Решение, какие роли нужны для выполнения тестирования, когда и как проводить тестирование и как оценивать результаты;
- Составление графика тестирования;
- Определение шаблонов для тестовой документации;
- Выбор метрик для мониторинга и контроля подготовки и проведения тестирования, исправления дефектов, проблем и рисков.

Критерии входа в тестирование определяют, когда нужно начинать тестирование. Примеры:

- Готовность и доступность тестового окружения;
- Готовность средства тестирования в окружении;
- Доступность тестируемого кода;
- Доступность тестовых данных.

Анализ и проектирование тестов

Это деятельность, во время которой общие цели тестирования материализуются в тестовые условия и тестовые сценарии.

Активности на данном этапе процесса тестирования:

- Рецензирование базиса тестирования:
  - функциональных и нефункциональных требований;
  - архитектуры;
  - дизайна;
  - технических требований к интерфейсу;
- Оценка тестируемости базиса тестирования и объектов тестирования;
- Идентификация и расстановка приоритетов тестирования;
- Выявление необходимых данных для поддержки тестовых условий и тестовых сценариев;
- Проектирование и установка тестового окружения и выявление необходимой инфраструктуры и инструментов;
- Создание двунаправленной трассируемости между тестовым базисом и

тестовым сценарием.

Внедрение и реализация тестов

Это деятельность, где процедуры тестирования или автоматизированные сценарии задаются последовательностью тестовых сценариев, а также собирается любая информация, необходимая для выполнения тестов, разворачивается окружающая среда, и запускаются тесты.

Активности на данном этапе процесса тестирования:

- Завершение, реализация и расстановка приоритетов тестовых сценариев (включая проектирование тестовых данных);
- Написание автоматизированных сценариев тестирования;
- Проверка правильности настройки тестового окружения;
- Проверка и обновление двунаправленной трассируемости между тестовым базисом и тестовым сценарием;
- Выполнение процедур тестирования либо вручную, либо используя инструменты выполнения тестов, согласно заданному плану;
- Регистрация результатов выполнения тестов;
- Сравнение фактических и ожидаемых результатов;
- Оформление отчетов об ошибках и занесение их баг-трекинговую систему;
- Повторное тестирование областей, где были исправлены ошибки и областей, где могут появится новые ошибки после исправления уже известных ошибок (регрессионное тестирование).

Оценка критериев выхода и создание отчетов

Это деятельность, где выполнение тестов оценивается согласно определенным целям. Она должна быть выполнена для каждого уровня тестирования.

Активности на данном этапе процесса тестирования:

- Сверка протокола тестирования в сравнении с критериями выхода, определенными в плане тестирования;
- Анализ необходимости использования дополнительных тестов или изменения критериев выхода;
- Написание итогового отчета о тестировании для заинтересованных лиц.

Критерии выхода определяют, когда нужно заканчивать тестирование.

Примеры:

- Степень покрытие кода, функциональности или рисков тестами;
- Оценку плотности дефектов или измерение надежности;
- Стоимость;
- Остаточные риски (неисправленные дефекты или недостаток тестового покрытия какой-либо области);
- План, основанный на времени выхода ПО на рынок.

9. Типы тестирования программного обеспечения (Статическое и динамическое, Ручное и автоматизированное)

Статическое тестирование проводится без исполнения кода. Сюда относится корректура, проверка, ревизия кода (при наблюдении за работой другого / парном программировании), критический анализ, инспекции и так далее.

Динамическое тестирование для получения корректных результатов требует исполнять код. Например, для модульных тестов, интеграционных, системных, приёмочных и прочих тестов. То есть тестирование проводится с использованием динамических данных, входных и выходных.

Ручное тестирование можно рассматривать как взаимодействие профессионального тестировщика и софта с целью поиска багов. Таким образом, во время ручного тестирования можно получать фидбек, что невозможно при автоматизированной проверке. Иными словами, взаимодействуя с приложением напрямую, тестировщик может сравнивать ожидаемый результат с реальным и оставлять рекомендации.

Автоматизированное тестирование — это написание кода. С его помощью ожидаемые сценарии сравниваются с тем, что получает пользователь, указываются расхождения. Автоматизированное тестирование играет важную роль в тяжёлых приложениях с большим количеством функций.

10. Методы тестирования программного обеспечения (Белого, черного, серого ящиков)

Метод "белого ящика"

Это тестирование, которое учитывает внутренние механизмы системы или компонента (ISO/IEC/IEEE 24765). Обычно включает тестирование ветвей, маршрутов, операторов. При тестировании выбирают входы для выполнения разных частей кода и определяют ожидаемые результаты. Традиционно тестирование белого ящика выполняется на уровне модулей, однако оно используется для тестирования интеграции систем и системного тестирования, тестирования внутри устройства и путей между устройствами.

Метод "черного ящика"

Это тестирования функционального поведения объекта (программы, системы) с точки зрения внешнего мира, при котором не используется знание о внутреннем устройстве тестируемого объекта. Под стратегией понимаются систематические методы отбора и создания тестов для тестового набора. Стратегия поведенческого теста исходит из технических требований и их спецификаций.

Метод "серого ящика"

Тестирование типа «серый ящик» проектируется со знанием программных алгоритмов и структур данных (белый ящик), но выполняется на пользовательском уровне (чёрный ящик). Сюда относится регрессионное тестирование и шаблонное тестирование (pattern testing).

11. Виды тестирования программного обеспечения (функциональное, нефункциональное, структурное, тестирование изменений, тестирование по приоритетам)

Функциональное тестирование

Тестирование, которое разрабатывается на основе функциональностей и возможностей системы и их взаимодействия со специфичными системами и могут быть выполнены на всех уровнях тестирования. Проводится методом «черного ящика».

Примеры:

- Позитивное тестирование;
- Негативное тестирование;
- Тестирование CRUD (Create, Read, Update, Delete);
- Тестирование по сценариям использования.

Нефункциональное тестирование

Тестирование, которое проводится для оценки характеристик систем и программ. Проверяется не корректность работы функций приложения, а сопутствующие характеристики.

Примеры:

- Тестирование внешнего вида приложения (методом «черного ящика»);
- Нагрузочное тестирование (методом «черного ящика» и методом «белого ящика»);
- Тестирование безопасности (методом «черного ящика»);
- Тестирование совместимости (методом «черного ящика»).

Структурное тестирование

Анализ и тестирование кода продукта, его архитектуры. Проводится методом «белого ящика».

Примеры:

- Unit-тесты;
- Интеграционные автоматизированные тесты;
- Тестирование веб-сервисов.

Тестирование изменений

Это повторное тестирование уже протестированных программ после внесения в них изменений, чтобы обнаружить дефекты, внесенные или пропущенные в результате этих действий. Чаще проводится методом «черного ящика».

Примеры:

- Регрессионное тестирование;
- Тестирование основанное на рисках.

Тестирование по приоритет

Виды тестирования, направленные на выявление качества функционала определенной важности.

- Дымовое тестирование (smoke test);
- Тестирование критического пути (critical path test);
- Расширенное тестирование (extended test).

12. Функциональное тестирование, Модели поведения пользователя, Позитивное и негативное тестирование

Функциональное тестирование

Тестирование, которое разрабатывается на основе функциональностей и возможностей системы и их взаимодействия со специфичными системами и могут быть выполнены на всех уровнях тестирования. Проводится методом «черного ящика».

Примеры:

- Позитивное тестирование;
- Негативное тестирование;
- Тестирование CRUD (Create, Read, Update, Delete);
- Тестирование по сценариям использования.

Виды функционального тестирования

- Позитивное тестирование;
- Негативное тестирование;
- Исследовательское тестирование;
- Интуитивное тестирование;
- Тестирование по сценариям использования (End-to-end testing);
- Тестирование основанное на ролях (Role-based testing);
- Инсталляционное тестирование;
- CRUD тестирование.

Позитивное тестирование - тестирование, при котором используются только валидные данные и выполняются только валидные действия

Негативное тестирование - тестирование с использованием невалидных данных и действий, направленное на получение ошибок и предупреждений

Пользователь-интуит

Пользователь не читал инструкций или не способен их прочитать. Как правило, это пользователи веб и мобильных приложений, находящихся в общем доступе. В процессе тестирования одинаковый приоритет отдается как позитивному, так и негативному тестированию. Также необходимо обращать внимание на несоответствие интерфейса/поведения программы существующим стереотипам.

«Хороший» пользователь

Добросовестный пользователь действует в строгом соответствии с инструкциями ПО. Главный приоритет отдается позитивному тестированию. Поиск ошибок осуществляется как в логике работы программы, так и в документации на программу.

«Плохой» пользователь

Недобросовестный пользователь стремится использовать программу непредусмотренным способом. Подобные пользователи чаще пользуются программами, которые содержат важную информацию о пользователе: данные банковских карт, стратегически важная информация для бизнеса, и т.д.

13. Исследовательское тестирование ПО (что это, отличие от сценарного тестирования, когда стоит и когда не стоит применять)

Исследовательское тестирование

“Исследовательское тестирование ПО – это стиль в тестировании ПО, который предполагает сочетание личной свободы тестировщика и его обязанности постоянно оптимизировать качество своей работы путем восприятия изучения ПО, проектирования тестов и самого тестирования, как взаимодополняемых активностей, которые выполняются одновременно на протяжении всей разработки ПО.”

Более простое определение исследовательского тестирования — это разработка и выполнения тестов в одно и то же время. Что является противоположностью **сценарного подхода** (с его предопределенными процедурами тестирования, неважно ручными или автоматизированными). Исследовательские тесты, в отличие от сценарных тестов, не определены заранее и не выполняются в точном соответствии с планом. Но при этом, даже в свободной форме поисковой сессии тест будет включать в себя ограничения состоящие в том, какую часть продукта тестировать или какую стратегию использовать.

Когда использовать:

- Мало времени
- Сложности с требованиями
- Небольшой проект
- Тестировщики проходят одни и те же сценарии при тестировании
- Отсутствие времени на тестировани при разработке новой фичи

Когда не надо(не лучшая практика) использовать:

- Большой проект
- Проводится интеграционное тестирование
- Приложение стандартизированное
- Тестовые сценарии отдаются на аутсорс

14. Исследовательское тестирование ПО (Туры, Туры по бизнес-центру)

Туры – это идеи и инструкции по исследованию программного продукта, объединенные определённой общей темой или целью.

Тур – это своего рода план тестирования, он отражает основные цели и задачи, на которых будет сконцентрировано внимание тестировщика во время сессии исследовательского тестирования. При этом тестировщик – это турист, а тестируемое приложение – это город. Обычно у туриста (тестировщика) мало времени, поэтому он выполняет конкретную задачу в рамках выбранного тура, ни на что другое не отвлекаясь. Город (ПО) разбит на районы: деловой центр, исторический район, район развлечений, туристический район, район отелей, неблагополучный район.

Деловой центр — это место, где делается бизнес. Как правило, это непривлекательный для туристов район, где сосредоточены банки, офисные здания. При исследовании ПО все наоборот. Деловой центр — это те функции, ради которых пользователи покупают и используют приложение. Это те killer-feature, которые описывают маркетологи, и которые упомянет любой из ваших пользователей при опросе, зачем им ваше приложение. Тур по деловому центру фокусирует внимание на главных частях вашего приложения и показывает сценарии их использования вашими клиентами.

Тур по путеводителю: тестирование по руководству пользователя

Денежный тур: найти функции, которые заставляют людей тратить деньги на ваше ПО и проверить их работу

Тур по ориентирам:

Метод Уиттакера, «дорога вперед» — наметить список ориентиров и исследовать приложение, переходя от одного к другому.

Метод Балто, «дорога назад» — сохранять свои действия, чтобы можно было вернуться к ним в любой момент. Делать закладки, сохранять настройки. Уходить далеко вглубь приложения, а потом возвращаться по закладкам.)

Интеллектуальный тур:

задавать приложению «сложные» вопросы, нагружать его по максимуму. Или наоборот, задавать глупые вопросы для привлечения внимания.

The FedEx Tour:

проследить за путем входных данных (inputs). Найти и проверить все функции, которые их используют: сохраняют, меняют, выводят (outputs).

Внеурочный тур: проверить все задачи, выполняющиеся в фоновом режиме.

Сборщик мусора:

Выбрать цели (например, все пункты меню, все сообщения об ошибках, все диалоговые окна) и посещать каждый пункт наикратчайшим путем. Не останавливаясь тестировать детально, но замечая очевидные вещи.

15. Исследовательское тестирование ПО (Туры, Туры историческому району, Туры по району развлечений)

Туры – это идеи и инструкции по исследованию программного продукта, объединенные определённой общей темой или целью.

Тур – это своего рода план тестирования, он отражает основные цели и задачи, на которых будет сконцентрировано внимание тестировщика во время сессии исследовательского тестирования. При этом тестировщик – это турист, а тестируемое приложение – это город. Обычно у туриста (тестировщика) мало времени, поэтому он выполняет конкретную задачу в рамках выбранного тура, ни на что другое не отвлекаясь. Город (ПО) разбит на районы: деловой центр, исторический район, район развлечений, туристический район, район отелей, неблагополучный район.

Туры по историческим районам

Исторические районы — части города, содержащие старые здания и достопримечательности. В ПО исторические районы могут быть слабо соединены или сосредоточены в одном месте. Исторические районы в ПО представляют собой: унаследованный код; функции, созданные в предыдущих версиях; исправления багов.

Бажные секции в коде надо тестировать особенно тщательно. Туры по историческим районам проверяют старую функциональность и исправления ошибок.

Тур по плохому району: Найти компонент в баг-трекере, в котором больше всего задач и исследовать его. Найти новое свойство или улучшение, с которым связано больше всего багов (если в баг-трекере расставляются связи между задачами) и исследовать его.

Музейный тур: Найти старый код в системе контроля версий, который: давно создан и больше не менялся; давно создан и недавно обновился. И протестировать функциональность, заложенную в код.

Тур предыдущей версии: функциональность, которая работала до новой версии, продолжает работать.

Туры по развлекательным районам

Туристы приходят в развлекательный район ради отдыха, а не достопримечательностей. Развлекательный район — функции для разметки страницы, форматирования, изменения фона. Другими словами, работа заключается в создании документа, а развлечение — в наведении красоты. Туры по развлекательным районам исследуют второстепенные и убеждаются, что они дополняют основные без противоречий.

Тур актера второго плана Проверить функции, которые не главные, но находятся рядом с ними.

Тур глухого переулка: проверить наименее используемые функции

Тур полуночника: Проверить, как долго приложение будет работать, оставаясь запущенным. Откройте приложение и не закрывайте его. Работайте с ним, нагружайте его, но не закрывайте. Всегда будет еще один клуб и последний бокал пива — помните эту заповедь тура, когда хочется все прекратить.

16. Исследовательское тестирование ПО (Туры, Туры по туристическому району, Туры по району отелей)

Туры – это идеи и инструкции по исследованию программного продукта, объединенные определённой общей темой или целью.

Тур – это своего рода план тестирования, он отражает основные цели и задачи, на которых будет сконцентрировано внимание тестировщика во время сессии исследовательского тестирования. При этом тестировщик – это турист, а тестируемое приложение – это город. Обычно у туриста (тестировщика) мало времени, поэтому он выполняет конкретную задачу в рамках выбранного тура, ни на что другое не отвлекаясь. Город (ПО) разбит на районы: деловой центр, исторический район, район развлечений, туристический район, район отелей, неблагополучный район.

Туры по туристическим районам

Туры по туристическим районам имеют несколько разновидностей. Это и короткие забеги для покупки сувениров, аналог кратких тест-кейсов для тестирования специфичных функций. Это и длинные поездки для посещения списка мест, которые хочется увидеть. Эти туры не о том, как заставить приложение работать, они о том, как посетить функциональность быстро… только чтобы сказать “мы тут были”!

Тур коллекционера Пройтись везде, где только можно, в приложении, и задокументировать все выходные данные. Собрать полную коллекцию.

Тур одинокого бизнесмена протестировать функции, которые расположены дальше всего от точки входа в приложение.

Тур супермодели Проверить, как приложение выглядит и какое первое впечатление производит.

Тур «Второй бесплатно» Проверить, как приложение работает в мультипоточном режиме. Запустите приложение, потом запустите вторую копию, потом третью. Теперь запустите на них функции, которые отжирают память, пишут на диск или блокируют файл.

Тур шотландского паба Найти и протестировать функции, которые сложно найти, если заранее о них не знаешь. Но с наскока простой пользователь их не найдет. Как житель мегаполиса пройдет мимо крутого паба, не зная, что внутри классное пиво и вежливые официанты.

Туры по отельным районам

Отель — убежище для туриста. Это место, куда можно сбежать из давки и суеты популярных мест для небольшого отдыха и расслабления. Сюда приходит тестировщик, уйдя от главной функциональности, чтобы потестировать второстепенные или сопутствующие основным фичам функции, которые часто игнорируются в тест-плане.

Тур, отмененный из-за дождя найти функции, которые работают дольше пары секунд, запустить и отменить!

Тур домоседа Лениться! Делать так мало работы, как только возможно:
соглашаться со всеми дефолтными значениями; оставлять поля пустыми;
заполнять в форме минимум значений; никогда не кликать на “подробнее

17. Исследовательское тестирование ПО (Туры, Туры по неблагополучному району)

Туры – это идеи и инструкции по исследованию программного продукта, объединенные определённой общей темой или целью.

Тур – это своего рода план тестирования, он отражает основные цели и задачи, на которых будет сконцентрировано внимание тестировщика во время сессии исследовательского тестирования. При этом тестировщик – это турист, а тестируемое приложение – это город. Обычно у туриста (тестировщика) мало времени, поэтому он выполняет конкретную задачу в рамках выбранного тура, ни на что другое не отвлекаясь. Город (ПО) разбит на районы: деловой центр, исторический район, район развлечений, туристический район, район отелей, неблагополучный район.

Туры по захудалым районам

Это непривлекательные места, о которых расскажет редкий путеводитель. Они полны мошенников и сомнительных личностей, и лучше обходить их стороной. Тем не менее, они привлекают определенный класс туристов.

Тур саботажника Саботировать работу приложения — запустить действие и отобрать ресурсы, необходимые для его выполнения.

Тур антисоциального типа

- opposite tour — вводить наименее хорошие входные данные везде, где только возможно — данные, которые выходят из контекста, явно глупые или полностью бессмысленные.

- illegal inputs — вводить значения, которые не должны быть введены. Входные данные неверного типа, неверного формата, слишком длинные, слишком короткие итд. Аналогично туру по плохим районам.

- wrong turn tour — выполнять действия в неправильном порядке. Возьмите группу правильных действий и перемешайте их так, чтобы последовательность оказалась неверной.

Обсессивно-компульсивный тур, или тур невротика

Повторять одно действие снова и снова. Вводить одинаковые значения, пока не надоест

Для тестировщика обязателен тур по этим районам для выявления тех опасностей, которые могут подстерегать пользователей продукта. Для тура отлично подойдут входные данные, ломающие приложение или способные каким-либо образом ему навредить.

18. Тестирование требований (Почему важно? Что тестируется? Параметры тестирования документации)

Требования – это первое, на что смотрит команда проекта, это фундамент для проектирования и разработки продукта. Допущенная в документации ошибка или неточность может проявиться в самый неподходящий момент. Очевидно, что гораздо проще устранить дефект в паре строк требований, чем позже переписать несколько сотен (или даже тысяч) строк кода.

Тестирование требований направлено на то, чтобы уже на начальных этапах проектирования системы устранить максимально возможное количество ошибок. В перспективе, это позволяет:

• значительно снизить итоговую стоимость проекта;
• улучшить качество продукта;
• сохранить нервы всей команде.

Тестируется – документация

Параметры:

- Корректные требования - Набор требований к программному обеспечению является корректным тогда и только тогда, когда каждое требование, сформулированное в нем, представляет нечто, требуемое от создаваемой системы.
- Недвусмысленные требования - Требование является недвусмысленным тогда и только тогда, когда его можно однозначно интерпретировать.
- Полнота набора требований - Набор требований является полным тогда и только тогда, когда он описывает все важные требования, интересующие пользователя, в том числе требования, связанные с функциональными возможностями, производительностью, ограничениями проектирования, атрибутами или внешними интерфейсами.
- Непротиворечивость набора требований - Множество требований является внутренне непротиворечивым, когда ни одно его подмножество, состоящее из отдельных требований, не противоречит другим подмножествам.
- Упорядочивание требований по их важности и стабильности - . Если ресурсы недостаточны, чтобы в пределах выделенного времени и бюджета реализовать все требования, очень полезно знать, какие требования являются не столь уж обязательными, а какие пользователь считает критическими.
- Проверяемость - Требование в целом является проверяемым, когда каждое из составляющих его элементарных требований является проверяемы, т.е. когда можно протестировать каждое из них и выяснить, действительно ли они выполняются.
- Модифицируемость - Множество требований является модифицируемым, когда его структура и стиль таковы, что любое изменение требований можно произвести просто, полно и согласованно, не нарушая существующей структуры и стиля всего множества.
- Трассируемость - Требование в целом является трассируемым, когда ясно происхождение каждого из составляющих его элементарных требований и существует механизм, который делает возможным обращение к этому требованию при дальнейших действиях по разработке.

19. Тестирование требований (Уровни и виды требований, Критерии качества требований, Методы тестирования требований)

Уровни и виды требований:

бизнес-треб,пользовательские треб,бизнес-правила, функциональные треб, нефункциональные треб, треб к интерфейсам,треб к данным

1. Бизнес-требования – выражают цель, ради кот. разрабатывается продукт. Результат выявления требований на этой уровне – это общее видение – документ, кот., как правило, представлен простым текстом и таблицами. Нет детализации поведения системы и иных технических хар-к, но мб определены приоритеты решаемых бизнес-задач, риски и т.д.
Исп. для проведения валидации и приемочного тестирования. Примеры:

- нужен инструмент в реальном времени, отображающий наиболее выгодный курс покупки и продажи валюты
- необх. в 2-3 раза повысить кол-во заявок, обрабат. 1 оператором за смену
- нужно автоматизировать процесс выписки товарно-транспортных накладных на основе договоров

2. Пользовательские требования – описывает задачи, кот. юзер может выполнять с пом. разрабатываемой системы (реакцию системы на д-вия юзера, операции работы юзера). Т.к. появляется описание поведения системы, требования этого уровня мб использованы для оценки объема работ, стоимости проекта, времени разработки и т.д.

- Варианты использования
- Пользовательских историй
- Пользовательских сценариев

Используются на системном уровне.

3. Бизнес-правила – описывают особенности принятых в предметной области (и/или непосредственно у заказчика) процессов, ограничений и иных правил. Эти правила могут относиться к бизнес-процессам, правилам работы сотрудников, нюансом работы ПО и т.д.

4. Функциональные требования – описывают поведение системы, т.е. ее действия (вычисления, преобразования, проверки, обработку...). В контексте проектирования функциональные требования в основном влияют на дизайн системы.

- В процессе инсталляции приложение должно проверять остаток свободного места на целевом носителе
- Система должна авто- выполнять резервное копирование данных ежедневно в указанный момент времени
- Эл. адрес юзера, вводимый при регистрации, должен быт проверен на соответствие требованиям RFC822

Используются часто на компонентном и интеграционном уровнях тестирования

5. Нефункциональные требования – описывают свойства системы (удобство использования, безопасность, надежность, расширяемость и т.д.), которыми она должна обладать при реализации своего поведения. Здесь проводится более техническое и более детальное описание атрибутов качества. В контексте

проектирования нефункц. требования в основном влияют на архитектуру системы. Используются на компонентном, интеграционном и системном уровнях. Используются для видов нефункционального тестирования.

- При одновременной и непрерывной работе в системе 1000 юзеров, минимальное время между возникновением сбоев дб более или равно 100 часов
- Ни при каких условиях общий объем используемой памяти не должен превышать 2 Гб
- Размер шрифта для любой надписи должен поддерживать настройку в диапазоне от 5 до 15 пунктов

6. Требования к интерфейсам – описывают особенности взаимодействия разрабатываемой системы с другими системами и ОС.

- Обмен д-ми между клиентской и серверной частями приложения при осущ. фоновых AJAX-запросов дб реализован в формате JSON
- Протоколирование событий должно вестись в журнале событий ОС
- Соединение с почтовым сервером должно выполняться согласно RFC3207 (“SMTP over TLS”)

7. Требования к данным – описывают структуру данных (и сами данные), являющиеся неотъемлемой частью разрабатываемой с-мы. Часто сюда относят описание БД и особенностей ее использования.

- Все данные системы должны храниться в БД
польз. документы – под управлением СУБД MongoDB
не польз. документы – под управлением СУБД MySQL
- Информация о кассовых транзакциях за текущий месяц должна храниться в операционной таблице, а по завершению месяца переноситься в архивную

Методы:

Метод просмотра (универсальный метод, выполняется бизнесаналитиком или тестировщиком):

- Ознакомление с требованиями;
- Проверка требований по критериям качества;
- Оформление дефектов в виде комментариев/вопросов.

Метод экспертизы (выполняется при участии команды из бизнесаналитиков, представителей заказчика, разработчиков, лояльных пользователей, тестировщиков):

- Планирование;
- Обзорная встреча;
- Подготовка;
- Совещание;
- Переработка и оформление изменений в требованиях;
- Завершающий этап.

Метод составления вариантов тестирования (выполняется тестировщиком). Варианты тестирования занимают промежуточную позицию между User Case и Test Case, помимо использования для тестирования требований в дальнейшем легко расширяются до Test Cases и составляют основу тестовой документации.

20. Нефункциональное тестирование (Что это, Его параметры, и виды. Инсталляционное тестирование)

Нефункциональное тестирование - это тестирование, которое проводится для оценки характеристик программного обеспечения. Проверяется не корректность работы функций приложения, а сопутствующие характеристики (надежность, скорость работы, совместимость с другим ПО или оборудованием, и т.д.).

Виды нефункционального тестирования

- Тестирование производительности и безопасности;
- Тестирование эргономичности (usability testing) и совместимости;
- UI тестирование;
- Тестирование локализации и интернационализации;
- A/B тестирование;
- Тестирование на отказ и восстановление;
- Тестирование на соответствие стандартам;
- Тестирование на прерывания (работы мобильного ПО);
- Тестирование соединения (работы мобильного ПО).

Нефункциональные типы тестирования

- Тестирование производительности
- Нагрузочное тестирование
- Тестирование на отказоустойчивость
- Тестирование совместимости
- Юзабилити-тестирование
- нагрузочное тестирование
- Тестирование на ремонтопригодность
- Тестирование масштабируемости
- Объемное тестирование
- Тестирование безопасности
- Тестирование аварийного восстановления
- Тестирование на соответствие
- Тестирование переносимости
- Тестирование эффективности
- Проверка надежности
- Базовое тестирование
- Тестирование на выносливость
- Тестирование документации
- Тестирование восстановления
- Интернационализация Тестирование
- Тестирование локализации

Инсталляционное тестирование/Тестирование установки- Проверяет, не возникает ли проблем при установке, удалении, а также обновлении программного продукта.

21. Нефункциональное тестирование (Тестирование производительности)

Нефункциональное тестирование - это тестирование, которое проводится для оценки характеристик программного обеспечения. Проверяется не корректность работы функций приложения, а сопутствующие характеристики (надежность, скорость работы, совместимость с другим ПО или оборудованием, и т.д.).

Тестирование производительности — это одна из сфер деятельности развивающейся в области информатики инженерии производительности, которая стремится учитывать производительность на стадии моделирования и проектирования системы, перед началом основной стадии кодирования.

Направления тестирования производительности:

- нагрузочное (load)
- стресс (stress)
- тестирование стабильности (endurance or soak or stability)
- конфигурационное (configuration)

Возможны два подхода к тестированию производительности программного обеспечения:

1.в терминах рабочей нагрузки: программное обеспечение подвергается тестированию в ситуациях, соответствующих различным сценариям использования;

2.в рамках бета-тестирования, когда система испытывается реальными конечными пользователями.

22. Нефункциональное тестирование (Тестирование эргономичности, Тестирование совместимости, Тестирование GUI)

Нефункциональное тестирование - это тестирование, которое проводится для оценки характеристик программного обеспечения. Проверяется не корректность работы функций приложения, а сопутствующие характеристики (надежность, скорость работы, совместимость с другим ПО или оборудованием, и т.д.).

Тестирование эргономичности (usability testing)

Исследование, выполняемое с целью определения, удобен ли некоторый искусственный объект (такой как веб-страница, пользовательский интерфейс или устройство) для его предполагаемого применения. Это метод оценки удобства продукта в использовании, основанный на привлечении пользователей в качестве тестировщиков, испытателей и суммировании полученных от них выводов.

Тестирование совместимости (compatibility testing)

Тестирование, целью которого является проверка корректной работы приложения в определенном окружении.

Может проверяться совместимость с:

- Аппаратная платформа;
- Сетевые устройства;
- Периферия (принтеры, CD/DVD-приводы, веб-камеры и пр.);
- Операционная система (Unix, Windows, MacOS, ...);
- Базы данных (Oracle, MS SQL, MySQL, ...);
- Системное программное обеспечение (веб-сервер, файрволл, антивирус, ...);
- Браузеры (Internet Explorer, Firefox, Opera, Chrome, Safari, и др.)

Тестирование графического пользовательского интерфейса

Тестирование, проверяющее соответствие внешнего вида продукта заявленным дизайнам и требованиям.

23. Нефункциональное тестирование (Тестирование глобализации, A/B тестирование, Тестирование на отказ и восстановление системы, Тестирование на соответствие стандартам)

Нефункциональное тестирование - это тестирование, которое проводится для оценки характеристик программного обеспечения. Проверяется не корректность работы функций приложения, а сопутствующие характеристики (надежность, скорость работы, совместимость с другим ПО или оборудованием, и т.д.).

• **Локализация ПО**

процесс адаптации ПО к культуре какой-либо страны. Как частность – перевод UI, документации и сопутствующих файлов ПО с одного языка на другой

• **Интернационализация ПО**

технологические приемы разработки, упрощающие адаптацию продукта к языковым и культурным особенностям региона, отличного от того, в кот. разрабатывался продукт

• **Отличие локализации от интернационализации:**

интернац – на нач. этапах разработки, локализ – для каждого целевого языка

Локализ: главный вопрос : “Все ли элементы страницы переведены?”

ошибка: название страницы не переведено, плейсхолдер, текст на картинках

свинина для евреев в онлайн-магазине

арабские эмираты: текст справа-налево, картинки детей и женщин

Интерн: специфический формат хранения д-х, универсальный для поддерживаемых регионов

дата, время (am/pm, 24), разделители в числах, суммах, валютах

в мобильных – счит. IP или системный язык в настройках

в браузерах – счит. браузерную локаль из системы

A/B тестирование

Метод маркетингового исследования, суть которого заключается в том, что контрольная группа элементов сравнивается с набором тестовых групп, в которых один или несколько показателей были изменены, для того, чтобы выяснить, какие из изменений улучшают целевой показатель.

Тестирование на отказ и восстановление системы

Тестирование, которое проверяет продукт с точки зрения способности противостоять и успешно восстанавливаться после возможных сбоев, возникших в связи с ошибками программного обеспечения, отказами оборудования или проблемами связи. Целью данного вида тестирования является проверка систем восстановления, которые, в случае возникновения сбоев, обеспечат сохранность и целостность данных тестируемого продукта.

Тестирование на соответствие стандартам

Процесс тестирования для определения соответствия компонента или системы стандартам, нормам и правилам.

24. Тестирование безопасности (Что это, Причины пробелов в безопасности, Тестирование проницаемости)

Тестирование безопасности - это процесс оценки системы, сети или веб-приложения с целью выявления уязвимостей, которые может использовать злоумышленник.

Причиной пробелов в безопасности могут быть различные факторы, такие как плохое проектирование или реализация, недостаток правильной настройки или обслуживания, или использование устаревшего или неподдерживаемого ПО. Кроме того, постоянно обнаруживаются новые угрозы безопасности, поэтому даже хорошо защищенные системы могут стать уязвимыми со временем.

Тестирование проникновения (penetration testing) - это тип тестирования безопасности, который имитирует атаку на систему, чтобы выявить и оценить уязвимости. Он включает в себя попытку получить несанкционированный доступ к системе или сети, чтобы выявить слабые места и оценить эффективность контролей безопасности системы.

25. Тестирование безопасности (Виды атак)

Есть множество различных типов атак, которые могут использоваться для эксплуатации уязвимостей безопасности в системе или сети. Некоторые из них включают:

Атаки на сеть: это атаки на инфраструктуру сети, которые могут включать в себя техники, такие как отказ в обслуживании (DoS) и распределенный отказ в обслуживании (DDoS) атаки.

Атаки на приложения: это атаки на определенные приложения или службы, которые могут включать в себя техники, такие как SQL-инъекция и кросс-сайтовая подделка скриптов (XSS).

Фишинг: это вид атак, который обычно использует техники социальной инженерии, чтобы обмануть пользователей, заставляя их раскрывать чувствительную информацию, например, учетные данные для входа

Атаки с использованием вредоносного ПО: это злонамеренное программное обеспечение, которое распространяется с целью нанести вред компьютерной системе, включая вирусы, черви, Трояны и шифровальщики.

Социальная инженерия: это не техническое использование психологии и поведения человека, обманывая людей нарушать процедуры безопасности или раскрывать чувствительную информацию.

Физические атаки: этот вид атаки включает в себя физическое уничтожение оборудования, кражу устройств и данных или нарушение энергоснабжения или систем охлаждения

Прослушивание/сниффинг: этот вид атаки перехватывает и мониторит трафик сети для кражи чувствительной информации.

Атаки "Man-in-the-middle" позволяют злоумышленнику перехватывать и читать или изменять сетевые коммуникации между двумя сторонами.

И тд

26. Тестирование безопасности (Виды вредоносного ПО)

Тестирование безопасности - это метод тестирования, позволяющий определить, защищена ли информационная система данными и поддерживает ли она функциональность по назначению. Тестирование безопасности не гарантирует полную безопасность системы, но важно включить тестирование безопасности как часть процесса тестирования.

Вредоносная программа (вредоносное ПО) - это любое программное обеспечение, которое частично контролирует систему для злоумышленника / создателя вредоносного ПО.

Malwares

Ниже перечислены различные виды вредоносного ПО:

Вирус - это программа, которая сама создает копии и вставляет эти копии в другие компьютерные программы, файлы данных или в загрузочный сектор жесткого диска. При успешной репликации вирусы вызывают вредную активность на зараженных компьютерах, например, кражу места на жестком диске или процессорного времени.

Червь - это тип вредоносного ПО, который оставляет свою копию в памяти каждого компьютера на своем пути.

Trojan - не самовоспроизводящийся тип вредоносного ПО, содержащий вредоносный код, который после выполнения приводит к потере или краже данных или возможному системному вреду.

Adware - рекламное ПО, также известное как бесплатное программное обеспечение или программное обеспечение для катания на потолке, представляет собой бесплатное программное обеспечение для компьютеров, которое содержит коммерческую рекламу игр, настольных панелей инструментов и утилит. Это веб-приложение, и оно собирает данные веб-браузера для таргетинга рекламных объявлений, особенно всплывающих окон.

Spyware - программное обеспечение для инфильтрации, которое анонимно контролирует пользователей, что позволяет хакеру получать конфиденциальную информацию с компьютера пользователя. Spyware использует уязвимости пользователей и приложений, которые нередко привязаны к бесплатной загрузке онлайн-программного обеспечения или к ссылкам, которые пользователи кликают.

Rootkit - это программное обеспечение, используемое хакером для доступа к уровню доступа администратора к компьютеру / сети, который устанавливается через украденный пароль или путем использования уязвимости системы без знаний жертвы.

27. Тестирование безопасности (Тестирование на проникновение)

Тестирование безопасности - это метод тестирования, позволяющий определить, защищена ли информационная система данными и поддерживает ли она функциональность по назначению. Тестирование безопасности не гарантирует полную безопасность системы, но важно включить тестирование безопасности как часть процесса тестирования.

Процесс тестирования безопасности

Тестирование безопасности можно рассматривать как контролируемую атаку на систему, которая реалистично раскрывает недостатки безопасности. Его цель - оценить текущий статус ИТ-системы. Он также известен как тест на проникновение или более популярным, как этический взлом.

Тест на проникновение проводится поэтапно. Надлежащая документация должна выполняться на каждом этапе, чтобы все этапы, необходимые для воспроизведения атаки, были доступны с готовностью. Документация также служит основой для получения подробного отчета, который клиенты получают в конце теста на проникновение.

Тест на проникновение включает четыре основные фазы:1. Сбор информации (footprinting),2. Сканирование,3.Перечисление,4.Эксплуатация

Footprinting - это процесс сбора плана конкретной системы или сети и устройств, которые подключены к рассматриваемой сети. Это первый шаг, который использует тестировщик проникновения для оценки безопасности веб-приложения.

Сканирование - это второй шаг. Он включает в себя сканирование открытых портов, отпечатки пальцев операционной системы и обнаружение служб на портах. Конечной целью сканирования является поиск открытых портов через внешнее или внутреннее сканирование сети, пинговые машины, определение сетевых диапазонов и сканирование портов на отдельных системах.

Перечисление. Цель перечисления - получить полную картину цели. На этом этапе тестировщик проникновения пытается определить действительные учетные записи пользователей или плохо защищенные общие ресурсы, используя активные подключения к системам.

Эксплуатация - это последний этап, когда тестировщик безопасности активно использует уязвимости безопасности, присутствующие в рассматриваемой системе. Как только атака будет успешной, можно проникнуть в другие системы в домене, потому что тестировщики проникновения затем имеют доступ к более потенциальным целям, которые раньше не были доступны.

28. Тестовая документация (Ее виды и назначение, Тест план)

Тестовая документация - это документация, создаваемая тестировщиками, которая помогает в выполнении различного рода активностей в рамках тестирования программного обеспечения.

Тестовая документация для планирования тестирования:

•Тест план
•График тестирования
•Матрица устройств
•Матрица прослеживаемости
•Тестовый набор
•Тест сценарии
•Тест кейсы
•Чеклист

Тестовая документация для отчетности:

•Отчеты об ошибках
•Отчеты о результатах тестирования

Тест план

Документ, описывающий весь объем работ по тестированию, начиная с описания тестируемых объектов, стратегии, расписания, критериев начала и окончания тестирования, до необходимого в процессе работы оборудования, специальных знаний, а также оценки рисков с вариантами их разрешения.

Цели написания тест плана:

•Продумать стратегию тестирования программного обеспечения;
•Описать процесс тестирования на проекте, и как он встраивается в процесс разработки;
•Обеспечить информированность каждого члена команды об активностях QA команды, распределении обязанностей и зон ответственности;
•Скорректировать ожидания заказчика от команды тестирования.

29. Тестовая документация (График тестирования, Матрица трассируемости, Тестовый набор, Чек-лист)

Тестовая документация - это документация, создаваемая тестировщиками, которая помогает в выполнении различного рода активностей в рамках тестирования программного обеспечения.

График тестирования: Документ описывающий последовательность выполнения активностей по тестированию членами QA команды, с указанием дат начала выполнения работ и их завершения.

Цели:

•Согласовать работу команды разработки и тестирования;

•Обеспечить информированность каждого члена команды о последовательности задач, а также о сроках их выполнения;

•Обеспечить прозрачность процесса тестирования для заказчика;

•Обеспечить возможность отслеживания отставаний от плана и влияния добавления дополнительных задач команде.

Матрица трассировки

Документ, используемый для определения покрытия требований проверками, оформленными в виде соответствующей тестовой документации.

Матрица представляет собой таблицу, где соотносятся ID требований с ID чеклистов, тест кейсов, тест сценариев и другой документации, которая используется на проекте. Цель - обеспечить должное покрытие всех функциональных и нефункциональных требований тестами.

Тестовый набор

Документ, вмещающий в себя набор тестов/тестовых случаев/тестовых сценариев. Тестовый набор может быть сформирован для определенного вида тестирования, уровня тестирования или для компонента с определенным приоритетом.

Процесс создания тестового набора:

1.Тестовые случаи в виде соответсвующих документов оформляются в системе управления тестовой документацией.

2.У каждого тестового случая проставляется label (отметка).

3.В соответствии с необходимостью, по определенному label фильтруются все проверки и в группу попадают только нужные.

Один тестовый случай может иметь несколько labels и может быть в нескольких тестовых наборах.

Чеклист-Документ, перечисляющий идеи для проверки. Документ, который очень поверхностно указывает, что необходимо проверить в приложении, но не указывает, как это сделать.

30. Техники тест дизайна (Что это, Методы черного ящика: классы эквивалентности, Анализ граничных значений)

Техники тест дизайна - это совокупность методов и процедур, которые используются для планирования, создания и исполнения тестов программного обеспечения. Они помогают разработчикам тестов определять, какие тесты необходимо написать и как их исполнять, чтобы максимально эффективно проверить функциональность программного обеспечения.

Методы черного ящика - это техники, которые используются для тестирования без доступа к исходному коду или к деталям реализации программного обеспечения. Они основаны на использовании входных данных и выходных данных для проверки функциональности.

Классы эквивалентности – один из методов который используется для тестирования, он позволяет группировать входные данные в несколько классов эквивалентности, используя критерии, такие как значения, типы или диапазоны. Это позволяет разработчикам тестов разделить входные данные на группы и разработать тест-кейсы для каждой группы.

Анализ граничных значений - это техника, которая используется для тестирования программного обеспечения на граничных значениях входных данных. Это позволяет найти ошибки, которые могут возникнуть при использовании минимальных или максимальных значений входных данных, или при использовании значений, которые находятся на границе допустимого диапазона. Так же он позволяет найти ошибки связанные с переполнением или недостаточным количеством ресурсов.

Эти техники дизайна теста помогают разработчикам тестов планировать, создавать и исполнять тесты, которые максимально эффективно проверяют функциональность программного обеспечения, и обеспечивают достоверность результатов тестирования.

31. Техники тест дизайна (Методы черного ящика: таблица решений, таблица (диаграмма) переходов)

Техники тест дизайна - это совокупность методов и процедур, которые используются для планирования, создания и исполнения тестов программного обеспечения. Они помогают разработчикам тестов определять, какие тесты необходимо написать и как их исполнять, чтобы максимально эффективно проверить функциональность программного обеспечения.

Методы черного ящика - это техники, которые используются для тестирования без доступа к исходному коду или к деталям реализации программного обеспечения. Они основаны на использовании входных данных и выходных данных для проверки функциональности.

Тестирование таблицы решений:

это техника, которая используется для тестирования программного обеспечения, которое основано на логических вычислениях и принимает решения на основе различных условий. Тестировщик использует таблицу, содержащую ряд условий и соответствующих действий, которые должны быть выполнены в зависимости от условий, и использует эту таблицу для создания тест-кейсов.

Таблица решений содержит триггерные условия, обычно комбинации значений «Истина» и «Ложь» для всех входных условий, и результаты действия для каждой комбинации условий. Каждый столбец таблицы соотносится с бизнес-правилом, определяющим уникальную комбинацию условий и результат выполнения действий, связанных с этим правилом.

Стандартом покрытия для таблицы тестирования решений обычно явл. наличие хотя бы одного теста для каждой колонки, что обычно включает в себя покрытие всех комбинаций триггерных условий.

Тестирование таблицы переходов:

Система может показывать различные отклики в зависимости от текущих условий или предшествовавшей истории состояний. Данный метод позволяет тестировщику рассм. систему с точки зрения ее состояний, переходов между состояниями, входов или событий, активизирующих изменения состояний (переходы) и действия, к которым приводят эти переходы. Состояния системы или тестируемого объекта разделяемы, определяемы и конечны.

Таблицы состояний демонстрирует связи между состояниями и входами и может подсказать возможные некорректные переходы.

Тестировщик использует эту таблицу для создания тест-кейсов, которые проверяют, что программное обеспечение переходит из одного состояния в другое корректно в зависимости от входных данных и событий

32. Техники тест дизайна (Методы черного ящика: тестирование по вариантам использования, попарное тестирование)

Техники тест дизайна - это совокупность методов и процедур, которые используются для планирования, создания и исполнения тестов программного обеспечения. Они помогают разработчикам тестов определять, какие тесты необходимо написать и как их исполнять, чтобы максимально эффективно проверить функциональность программного обеспечения.

Методы черного ящика - это техники, которые используются для тестирования без доступа к исходному коду или к деталям реализации программного обеспечения. Они основаны на использовании входных данных и выходных данных для проверки функциональности.

Варианты использования (Use Case Testing)

"Тестирование по вариантам использования" (use case testing) - это метод, который заключается в тестировании системы в соответствии с различными сценариями использования, которые могут возникнуть в реальной жизни.

Use case — это сценарии, описывающие то как actor (обычно человек, но может быть и другая система) пользуется системой для достижения определенной цели. Варианты использования описываются с точки зрения пользователя, а не системы. Внутренние работы по поддержанию работоспособности системы не являются частью варианта использования.

Хотя бы один тест-кейс должен проверять основной сценарий и хотя бы по одному кейсу должно приходится на альтернативные сценарии.

Попарное тестирование

"Попарное тестирование" (pairwise testing) - это метод, который заключается в тестировании всех возможных комбинаций входных значений для двух или более параметров, которые могут влиять на работу системы.

Если количество комбинаций значений переменных велико, не стоит пытаться протестировать все возможные комбинации, лучше сосредоточиться на тестировании всех пар значений переменных.

Два подхода попарного тестирования (pairwise testing): метод ортогонального массива (orthogonal arrays) и метод всех пар (allpair algorithm).

Ортогональный массив — это двумерный массив, обладающий особым свойством: если выбрать две любые колонки в массиве, то в них будут присутствовать все возможные сочетания значений параметров, тем же самым свойством обладают все пары колонок.

Все пары — для создания массива используется алгоритм, генерирующий пары напрямую, без использования дополнительной балансировки. Если имеется большое количество параметров, принимающих маленькое количество значений, то для составления пар лучше использовать этот метод.

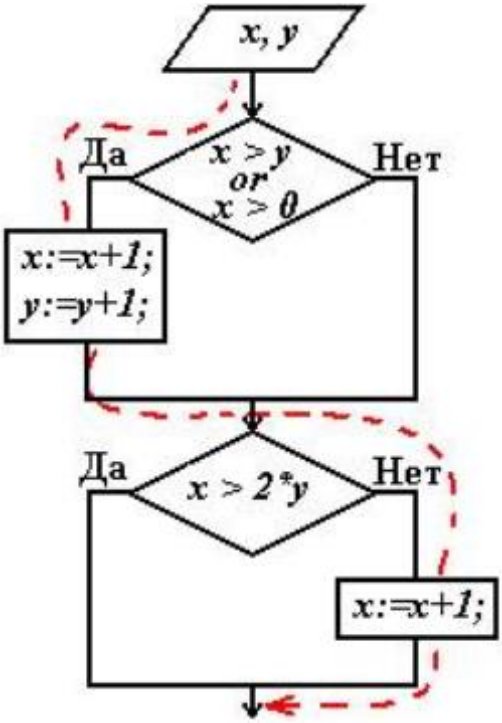
Не обязательно составлять попарные комбинации вручную, для этого существует масса инструментов.

Нужно учитывать, что могут возникнуть ограничения связанные с тем, что некоторые сочетания параметров никогда не будут иметь места.

33. Техники тест дизайна (Методы белого ящика: тестирование покрытия операторов, тестирование покрытия ветвей. Методы, основанные на опыте)

Техники тест дизайна - это совокупность методов и процедур, которые используются для планирования, создания и исполнения тестов программного обеспечения. Они помогают разработчикам тестов определять, какие тесты необходимо написать и как их исполнять, чтобы максимально эффективно проверить функциональность программного обеспечения.

"Белый ящик" - это метод тестирования, в котором особое внимание уделяется внутреннему действию продукта, а не его внешнему виду и поведению



Метод покрытия операторов.

Этот метод требует написания такого количества тестов, чтобы при выполнении их всех каждый оператор был выполнен хотя бы один раз.

Рассмотрим для примера постенькую блок-схему

Метод покрытия решений (путей).

Метод покрытия решений требует такого количества тестов, чтобы при выполнении их всех по каждой траектории, соединяющей соседние элементы блок-схемы вычисление прошло хотя бы один раз. Это означает, что каждое решение должно принимать как истинные, так и ложные значения. Именно это обеспечивает использование всех путей, выходящих из точек ветвления. Что касается операторов выбора, к ним это также относится. Для того, чтобы сохранит общность рассуждений, надо только переделать их (мысленно) в эквивалентную цепочку условных операторов.

Методы, основанные на опыте (Experience-based Test Techniques)

это метод тестирования, который заключается в использовании опыта и знаний тестировщика, чтобы предсказать и идентифицировать возможные проблемы в продукте.

- Предположение об ошибках (Error Guessing). способ предотвращения ошибок, дефектов и отказов, основанный на знаниях тестировщика.
- Исследовательское тестирование (Exploratory Testing). тест-кейсы и чек-листы создаются, выполняются, анализируются и оцениваются динамически во время выполнения тестов. подходит в ситуациях, когда документация недостаточная, либо вовсе отсутствует, в условиях очень сжатых сроков и как дополнение к другим, более формальным, методам тестирования.
- Тестирование на основе чек-листов (Checklist-based Testing). тестировщик проектирует, реализует и выполняет тесты, указанные в чек-листе.

Такие списки могут быть построены на опыте, на исторических данных об ошибках, на информации о приоритетах для пользователей и понимании, как и почему происходят отказы в программе.

34. Тестовые случаи (Цели, Жизненный цикл, Атрибуты тестовых случаев)

Тестовый случай – формально описанный алгоритм тестир программы, специально созданный для определения возникновения в программе определенной ситуации, определенных выходных данных.

Цель написания test case-ов:

- Подготовиться к тестированию (осн. цель)
- Детально описать шаги тестир и ожидаемый результат
- Задокументировать требования
- Облегчить передачу знаний по проекту
- Подготовиться к автоматизации тестирования

Атрибуты test case-а: (обяз. поля)

- **ID** – tc№
- **Краткое название тест случая**
- **Цель тестового случая** – что проверяем и при каких усл
- **Предусловия (Precondition)** – нумерованный список шагов чтобы добраться до компонента (ожидаемый резудьтат прописывать не надо)
- **Шаги (Steps)** – опис тестового действия, напр на проверку выбран требов)
- **Ожидаемый результат (Expected Result)** – пропис к каждому шагу, должен быть подробным и однозначным
- **Постусловия (Postcondition)** – д-вия, чтобы вернуть систему в исх сост
- **Статус (Status)**
- **Уровень (Level)** – выставл в соотв с уровнем тестир, для к-го он предназн: (компонентный, интеграционный, системный, приемочный)
- **Приоритет (Priority)**
- **Автор (Author)** – в с-мах tc management выставляется авто-
- **Комментарии (Notes, Comments)**

Жизненный Цикл:

серия этапов, которые проходит каждый тестовый случай от его создания до окончания его использования

Создан -> Запланирован(не выполнен->требует доработки) ->Выполняется(пропущен ->требует доработки) -> Пройден успешно (провален/заблокирован -> требует доработки) -> Закрыт. И снова по кругу

35. Тестовые случаи (Свойства качественных тестовых случаев, типичные ошибки при создании тестовых случаев)

Тестовый случай – формально описанный алгоритм тестир программы, специально созданный для определения возникновения в программе определенной ситуации, определенных выходных данных.

Правила написания тестовых случаев(свойства):

• Один тестовый случай должен проверять только одно требование. Допустимо объединять одно функциональное требование со связанными с ним требованиями к внешнему виду (например, когда речь идет о внешнем виде сообщения об ошибке). Но проверять два функциональных требованиям в одном тестовом случае не допустимо.

• Количество шагов не должно превышать 5-7. Если шагов больше, то возможно в тестовом случае проверяется несколько требований и его нужно разделить;

• В Steps / Ожидаемом результате должно быть описано только то, что относится к цели тест кейса;

• Шаги, которые не относятся к цели тест кейса – это либо Предусловие, либо Постусловие;

• Желательно располагать тестовые случаи таким образом, чтобы тестировщик "двигался" последовательно по приложению, максимально уменьшая количество повторяющихся действий;

• Для повышения читаемости и поддержки тестовые случаи должны быть разделены на смысловые части (как правило, по компонентам);

• В тестовых случаях необходимо избегать ссылок на сторонние документы, так как это уменьшает удобство работы с проверками и приложением одновременно.

Ошибки:

- Абстрактные названия
- Повелительное наклонение
- Указание ссылки для тестирования продукта на продакшене
- Слишком детализированное описание
- Отсутствие нужной информации (к примеру для авторизации и тд.)
- Плохое описание проверки (Итогового результата)

36. Тестовые сценарии (Цели, Шаги для создания, Атрибуты тестовых сценариев)

Тест. сценарии – формально описанный алгоритм тестир программы, специально созд. для опред. возникновения в проге опред. ситуации, опред. выходных д-х

Цель написания тест. сценариев:

подготовиться к проведению след. видов тестир:

* end-to-end
* explorary testing
* role-based testing

Атрибуты тест. сценария:

* Идентификатор (ID)
* Краткое название (Summary, Title)
* Описание сценария (Description): 1-5 предл
* Участники (Primary Actors): роли (юзеры), кот. фигурир в Тестовом Сценарии
* Предусловия (Precondition): подгот. с-му к вып. Сценария
* Осн. сценарий (Basic Flow)
* Альтернатив. сценарий (Alternative Flow)
* Исключения (Exceptional Flow)
* Статус (Status): как у Тест Кейса
* Приоритет (Priority): как у ТК (высший, средний, низкий)

Правила создания хороших сценарных тестов:

1) запрет использования для генерации начальных данных
2) тесты должны состоять из частей
3) они должны быть независимы от окружения
4) они должны использовать параметры
5) использовать код только для спец случаев
6) проверять результат выполнения

37. Тест план, его назначение и структура

Тест стратегия – офиц. док, опис. методологию тестир, принятую в компании. Одна и та же организация м. иметь разные стратегии для разных продуктов, рзных циклов разработки, разных уровней риска.

Виды тест стратегий:

- ✓ Аналитические стратегии- тестир, осн. на рисках (risk-based testing) - тестир, осн. на требованиях (requirements-based testing)
- ✓ Стратегии, осн на моделях - тестир, осн на моделях использ приложения
- ✓ Методические стратегии - тестир по общепринятым стандартам (ISO 25010:2011)
- ✓ тестир по стандартам, принятым в компании
- ✓ Стратегии, соотв процессуальным нормам - тестир по стандартам HIPPA, GDPR и т.д.
- ✓ Реактивные стратегии
- ✓ исслед тестир
- ✓ Консультативные стратегии
- ✓ тестир, осн на сценариях и д-х, предост заказчиком ПО
- ✓ Стратегии, искл регрессионное (повторное) тестир
- ✓ широкое использ автоматизации для любых повтор. тестов

Тест стратегия вкл:

- ✓ Описание процесса интеграции тестир в процессе разработки
- ✓ Техники тест дизайна
- ✓ Методы и виды тестир
- ✓ Уровни тестир
- ✓ Обяз и необяз стандарты, к-рым д. соотв ПО
- ✓ Критерии начала и окончания тестир
- ✓ Метрики, собираемые в процессе тестир
- ✓ Инструменты, исп в тестир
- ✓ Окружение, где проходит тестир
- ✓ Процесс контроля качества и его метрики
- ✓ Дефект менеджмент
- ✓ Роли и обяз-сти членов команды тестир

38. Отчеты об ошибках (Цель, Жизненный цикл, Атрибуты)

Отчет об ошибках – доказ, которое описывает и приоритизир. обнаруженный дефект, а также содействует его устранению.

Дефект – любое отклонение фактич результата от ожиданий пользователя, сформирован на основе требований, спецификаций, иной документации или опыта и здравого смысла. *Разработчик не имеет права закрывать дефект! (только тестировщик)*

Цель отчета об ошибках:

✓**Предоставить инфу о проблеме:** уведомить проектную команду и иных заинтересованных лиц о наличии проблемы, описать суть проблемы

✓**Приоритезировать проблему:** опред степень опасности и желаемые сроки ее устранения

✓**Содейств устранению проблемы:** предост необх. подробности для понимания сути случившегося, а также анализ причин возник проблемы и рекомендации по исправл ситуации

Жизненный цикл бага:

Жизненный цикл отчета об ошибке включает в себя следующие этапы: обнаружение ошибки, отправка отчета об ошибке, анализ ошибки, исправление ошибки, тестирование и закрытие отчета.

Атрибуты отчета об ошибках:

✓**ID**
✓**Краткое описание**
✓**Описание : Шаги воспроизв, Актуальный, Ожидаемый рез.**
✓Воспроизводимость
✓**Важность (Severity)**
✓**Срочность (Priority)**
✓возможность обойти
✓комментарий
✓**среда воспроизв**
✓компонент
✓тег
✓**ответственный**
✓приложения к багу
✓срок исправления бага
✓версия билда, где был обнаружен баг
✓версия, в кот. дб исправлен баг
✓**создатель отчета об ошибке (авто-)**
✓оценка трудозатрат на исправления бага

39. Отчеты об ошибках (Свойства качественных отчетов об ошибках, Типичные ошибки при написании)

Свойства качественных отчетов об ошибках включают:

- Полноту: отчет должен быть достаточно детальным и содержать всю необходимую информацию для идентификации и исправления ошибки.
- Ясность: отчет должен быть понятным и легко читаемым для того, кто будет его анализировать и исправлять.
- Конкретность: отчет должен содержать конкретные детали ошибки, такие как код, скриншоты и сообщения об ошибках.

Типичные ошибки, которые могут быть сделаны при написании отчета об ошибке, включают:

- Недостаточно информации: не предоставление достаточной информации для идентификации и исправления ошибки
- Неоднозначность: использование неоднозначной или непонятной терминологии
- Неконкретность: отсутствие конкретных деталей ошибки, таких как код, скриншоты и сообщения об ошибках
- Необоснованность: не дание достаточных обоснований для заявленной ошибки
- Неправильное форматирование: неправильное форматирование или оформление отчета, которое может затруднять чтение и анализ
- Неактуальность: предоставление неактуальной информации или отчет об ошибке, которая уже была исправлена.

ОШИБКИ:

- Ошибки оформления и формулировок

1.Плохие краткие описания
2.Идентичные краткие и подробные описания
3.Отсутствие в подробном описании явного указания фактического результата
4.Игнорирование кавычек, приводящее к искажению смысла
5.Лишние пункты в шагах воспроизведения
6.Пунктуационные, орфографические, синтаксические и им подобные ошибки

- Логические ошибки

1.Выдуманные дефекты
2.Отнесение расширенных возможностей приложения к дефектам.
3.Чрезмерно заниженные (или завышенные) важность и срочность
4.Концентрация на мелочах в ущерб главному.
5.Техническая безграмотность.
6.Игнорирование «последовательных дефектов

Это основные ошибки, которые могут быть сделаны при написании отчета об ошибке, но не являются полным списком. Важно, чтобы отчеты были полными, ясными и конкретными, чтобы они могли быть легко анализированы и исправлены.

40. Отчеты о результатах тестирования

Отчёт о результатах тестирования — документ, обобщающий результаты работ по тестированию и содержащий информацию, достаточную для соотнесения текущей ситуации с тест-планом и принятия необходимых управленческих решений.

К низкоуровневым задачам отчётности в тестировании относятся:

•оценка объёма и качества выполненных работ;

•сравнение текущего прогресса с тест-планом

•описание имеющихся сложностей и формирование рекомендаций по их устранению;

•предоставление лицам, заинтересованным в проекте, полной и объективной информации о текущем состоянии качества проекта, выраженной в конкретных фактах и числах.

•Информативность

•Точность и объективность

В общем случае отчёт о результатах тестирования включает следующие разделы:

•**Краткое описание.**В предельно краткой форме отражает основные достижения, проблемы, выводы и рекомендации.

•**Команда тестировщиков**. Список участников проектной команды, задействованных в обеспечении качества, с указанием их должностей и ролей в подотчётный период.

•Описание процесса тестирования

•Расписание работы команды тестировщиков и/или личные расписания участников команды.

•Статистика по новым дефектам

•Список новых дефектов с краткими описаниями и важностью.

•Статистика по всем дефектам.Таблица, график, отражающий такие статистические данные.

•Рекомендации (recommendations). Обоснованные выводы и рекомендации по принятию тех или иных управленческих решений (изменению тест-плана, запросу или освобождению ресурсов и т.д.)

•Приложения (appendixes). Фактические данные (как правило, значения метрик и графическое представление их изменения во времени).

41. Автоматизация тестирования (Виды, Модульное тестирование)

Автоматизированное тестир – часть процесса тестир на этапе контроля кач-ва в процессе разработки ПО. Оно использует программные средства для выполнения тестов и проверки рез-тов выполнения, что помогает сократить время тестир и упростить его процесс

Относится к **функциональному** виду тестир и проверяет корректность работы отдельных методов/компонентов/всего приложения

Виды авто-тестир:

* уровень модульного тестир (unit tests)
* уровень функционального тестир (non-UI-tests)
* уровень тестир через польз. интерфейс (UI tests)

Модульное тестир – процесс в программировании, позволяющий проверить на корректность отдельные модули исх. кода программы

Идея сост. в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное измненение кода к регрессии, т.е. к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

Правила UNIT-тестов

Тесты должны быть:

* достоверными
* не зависеть от окружения, на кот. они выполняются
* легко поддерживаться
* легко читаться и быть простыми для пониманияс (даже новый разработчик д. понять что именно тестируется)
* соблюдать единую конвенцию именования
* запускаться регулярно в автоматическом режиме
* один тест должен проверять только одну сущность
* тесты должны загружаться в с-ме контроле версий
* названия должно быть «говорящими»

42. Автоматизация тестирования (Интеграционные тесты, UI тестирование)

Автоматизированное тестир – часть процесса тестир на этапе контроля кач-ва в процессе разработки ПО. Оно использует программные средства для выполнения тестов и проверки рез-тов выполнения, что помогает сократить время тестир и упростить его процесс

Интеграционное тестирование – это тип тестирования, при котором программные модули объединяются логически и тестируются как группа. Как правило, программный продукт состоит из нескольких программных модулей, написанных разными программистами. Целью нашего тестирования является выявление багов при взаимодействии между этими программными модулями и в первую очередь направлен на проверку обмена данными между этими самими модулями.

Юнит-тест удовлетворяет следующим трем требованиям:
проверяет правильность работы одной единицы поведения;
делает это быстро;
и в изоляции от других тестов.
Тест, который не удовлетворяет хотя бы одному из этих трех требований, относится к категории интеграционных тестов.

С другой стороны, интеграционные тесты проходят через больший объем кода, что делает их более эффективными в защите от багов по сравнению с юнит-тестами. Они также более отделены от рабочего кода, а, следовательно, обладают большей устойчивостью к его рефакторингу.

ВИДЫ ПОДХОДОВ К ИНТЕГРАЦИОННОМУ ТЕСТИРОВАНИЮ

Большой взрыв , Снизу вверх, Сверху вниз ,Смешанный / сэндвич

UI-тестир – процесс в программировании, позвол. проверить работоспос-сть и внешний вид приложения, используя графический интерфейс приложения.

Инструмент -> Браузер -> Сервер

Во время данного тестирования тест полностью имитирует работу пользователя (открывается браузер, нажимаются кнопки, вводятся данные). При этом, если элемент не догрузился или отсутствует на странице, то действие не мб выполнено. Это делается специально, для того, чтобы предотвратить ошибки когда у реального пользователя есть баг, а тест его не находит.

Инструменты – VisualStudio

Подходы в UI автоматизации

Driven означает, что тесты зависят от каких-то определенных обстоятельства

- data-driven подход (данные)(используется в приложениях зависящих от введенных данных))

keyword-driven подход (ключевые слова) (Часто используется, когда команда автоматизации готовит карту объекта, а команда ручных разработчиков составляет тест так, как им нужно. Тест строится с помощью набора объектов.)

- behavior-driven подход (поведение) (подход, который зависит от поведения системы)