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 при использовании)

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)

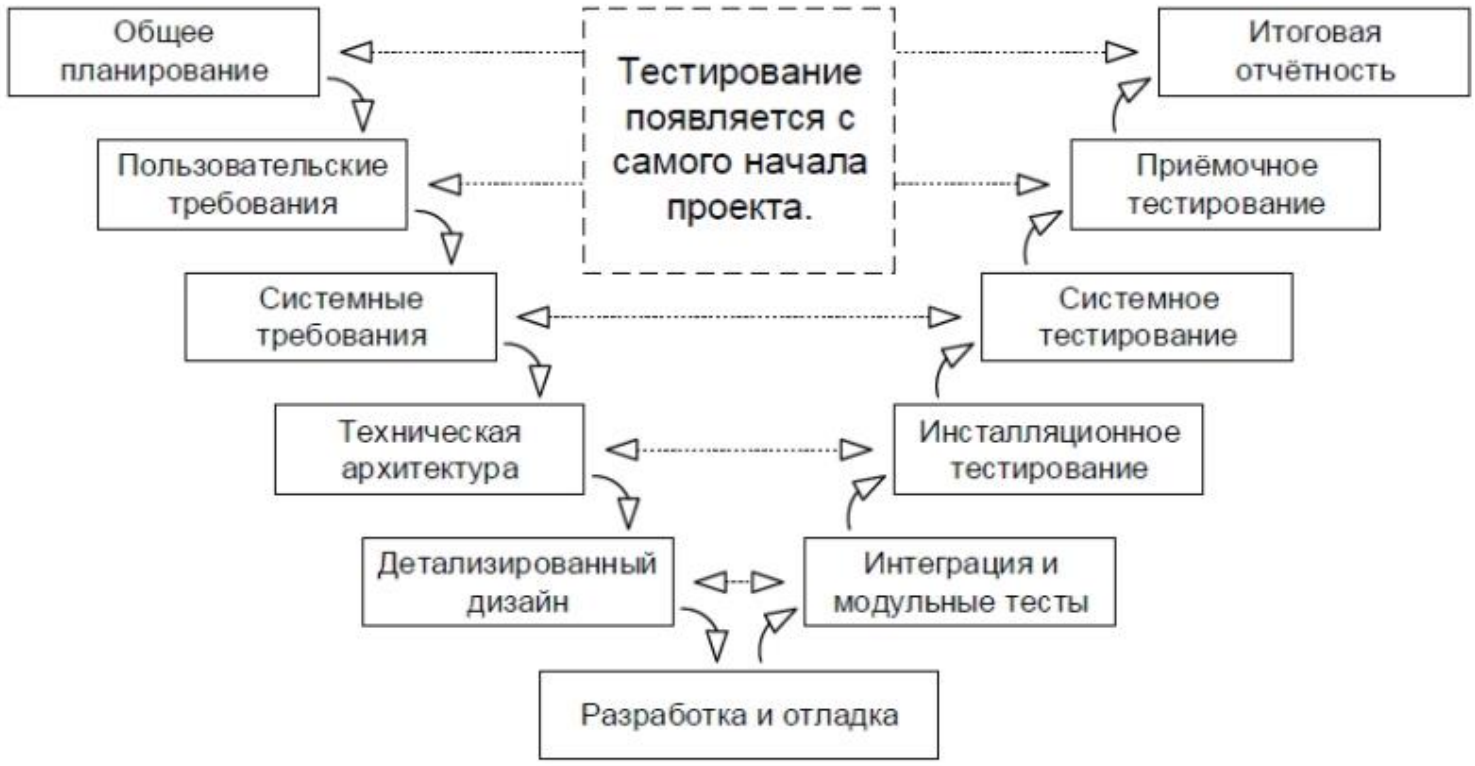
Водопадная модель (waterfall model) в современных проектах практически неприменима. Она предполагает однократное выполнение каждой из фаз проекта, которые, в свою очередь, строго следуют друг за другом.

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-model) является логическим развитием водопадной. Как водопадная, так и v-образная модели жизненного цикла ПО могут содержать один и тот же набор стадий, но принципиальное отличие заключается в том, как эта информация используется в процессе реализации проекта.



Очень упрощённо можно сказать, что при использовании v-образной модели на каждой стадии «на спуске» нужно думать о том, что и как будет происходить на соответствующей стадии «на подъёме». Тестирование здесь появляется уже на самых ранних стадиях развития проекта, что позволяет минимизировать риски, а

также обнаружить и устранить множество потенциальных проблем до того, как они станут проблемами реальными.

На практике v-образная модель может иметь большее или меньшее количество уровней разработки и тестирования, все зависит от конкретного проекта и разрабатываемого продукта.

Итерационная инкрементальная модель (iterative model, incremental model) является фундаментальной основой современного подхода к разработке ПО. Как следует из названия модели, ей свойственна определённая двойственность (а ISTQB-глоссарий даже не приводит единого определения, разбивая его на отдельные части):

- с точки зрения жизненного цикла модель является итерационной, т.к. подразумевает многократное повторение одних и тех же стадий;
- с точки зрения развития продукта (приращения его полезных функций) модель является инкрементальной.

Ключевой особенностью данной модели является разбиение проекта на относительно небольшие промежутки (итерации), каждый из которых в общем случае может включать в себя все классические стадии, присущие водопадной и v-образной моделям. Итогом итерации является приращение (инкремент) функциональности продукта, выраженное в промежуточном билде.

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.

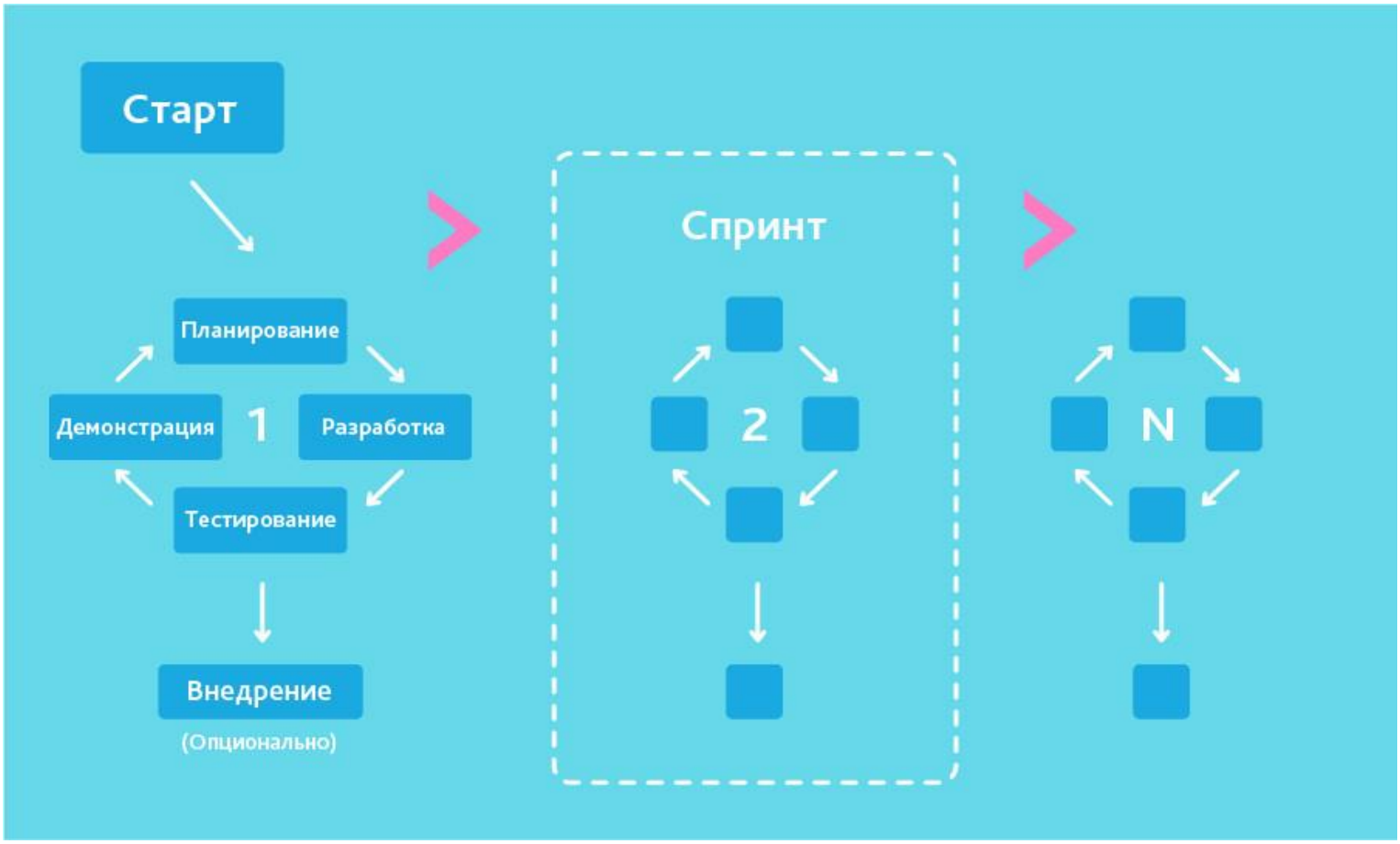
Спиральная модель предполагает 4 этапа для каждого витка:

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

Она хорошо работает для решения критически важных бизнес-задач, когда неудача несовместима с деятельностью компании, в условиях выпуска новых продуктовых линеек, при необходимости научных исследований и практической апробации.

Эта модель не подойдет для малых проектов, она резонна для сложных и дорогих, например, таких, как разработка системы документооборота для банка, когда каждый следующий шаг требует большего анализа для оценки последствий.

Гибкая модель (agile model) представляет собой совокупность различных подходов к разработке ПО и базируется на т.н. «agile-манифесте»:



В «гибкой» методологии разработки после каждой итерации заказчик может наблюдать результат и понимать, удовлетворяет он его или нет. Это одно из преимуществ гибкой модели. К ее недостаткам относят то, что из-за отсутствия конкретных формулировок результатов сложно оценить трудозатраты и стоимость, требуемые на разработку. Экстремальное программирование (XP) является одним из наиболее известных применений гибкой модели на практике.

В основе такого типа — непродолжительные ежедневные встречи — «**Scrum**» и регулярно повторяющиеся собрания (раз в неделю, раз в две недели или раз в месяц), которые называются «Sprint». На ежедневных совещаниях участники команды обсуждают:

- отчёт о проделанной работе с момента последнего Scrum'а;
- список задач, которые сотрудник должен выполнить до следующего собрания;
- затруднения, возникшие в ходе работы.

Методология подходит для больших или нацеленных на длительный жизненный цикл проектов, постоянно адаптируемых к условиям рынка.

Когда использовать Гибкую мод:

- Когда потребности пользователей постоянно меняются в динамическом бизнесе.
- Изменения на Гибкой мод. реализуются за меньшую цену из-за частых инкрементов.
- сложность её применения к крупным проектам
- частое ошибочное внедрение её подходов, вызванное недопониманием фундаментальных принципов модели.

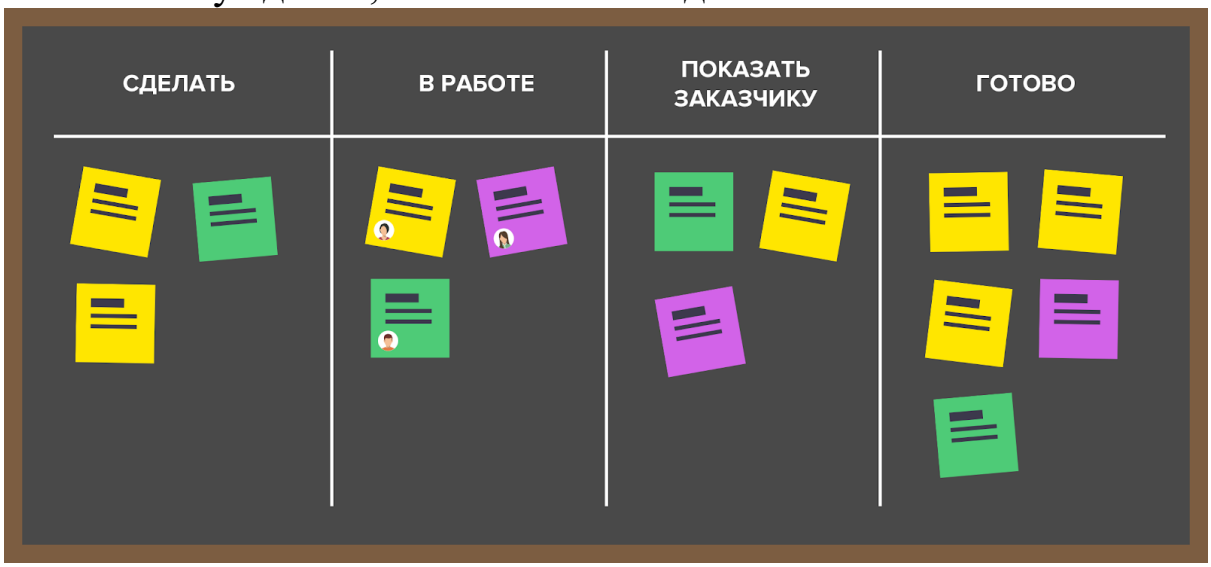
В отличие от модели водопада, в гибкой модели для старта проекта достаточно лишь небольшого планирования.

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

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.

При этом принцип формирования каждого столбца должен быть один. Например, это могут быть этапы производственного процесса («прототипирование», «дизайн», «разработка», «тестирование») или статусы выполнения задач («предстоит сделать», «в работе», «на проверке», «завершено»). По каждой колонке должно быть определено ограничение объема незавершенной работы — это позволяет предупредить перегрузы и простои. Этот принцип берет свое начало в законе американского ученого Джона Литтла, согласно которому при увеличении количества одновременно выполняемых задач, снижается скорость выполнения каждой из них. Поэтому команды постоянно балансируют между ограничением на невыполненную работу и скоростью пропускной системы. Лучшие практики ведения канбан-доски основаны на простых компонентах — обсуждение, баланс и взаимодействие.



Экстримальное программирование - XP:

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.

Ряд перечень основных тезисов-компонентов:

- *План релизов.* План релизов определяет даты релизов и формулировки пользователей, которые будут воплощены в каждом из них.

- *Планирование итераций* начинается со встречи в начале каждой итерации с целью выработки плана шагов для решения программных задач. Каждая итерация должна длиться от одной до трех недель. Формулировки внутри итерации сортируются в по-рядке их значимости для заказчика. Кроме того, добавляются задачи, которые не смогли пройти тесты приемки и требуют доработки. Формулировки и результаты тестов переводятся в программные задачи. Задачи записываются на карточках, которые образуют детальный план итерации. Для решения к каждой из задач требуется от одного до трех дней. Задачи, для которых нужно менее одного дня, можно сгруппировать вместе, а большие задачи разделить на несколько мелких. Разработчики оценивают задачи и сроки, для их выполнения.
- *Stand Up.* Каждое утро проводится собрание для обсуждения проблем, их решений и для усиления концентрации команды. Собрание проводится стоя для избежания длительных дискуссий не интересных всем членам команды. Ну тут дефолт, подробно расписывать не буду.
- *Простота.* Простой дизайн всегда занимает меньше времени, чем сложный. Поэтому всегда делайте самые простые вещи, которые только смогут работать.
- *Система метафор.* Выбор системы метафор нужен, чтобы удержать команду в одних и тех же рамках при именовании классов и методов. То, как вы называете свои объекты, очень важно для понимания общего дизайна системы и повторного использования кодов.
- *Заказчик на рабочей площадке*Это участие заказчика в процессе разработки. Если привлечь заказчика или его представителя не удается, иногда оказывается целесообразным временный наем специалиста в разрабатываемой области.
- *Тестирование до начала разработки.* В экстремальном программировании роль тестирования интереснее: теперь вначале идет тест, а потом код.
- *Парное программирование.*
- *Смена позиций. Во время очередной итерации всех работников следует перемещать на новые участки работы.*
- *Коллективное владение кодом.* Коллективное владение кодом стимулирует разработчиков подавать идеи для всех частей проекта, а не только для своих модулей. Любой разработчик может изменять любой код для расширения функциональности и исправления ошибок.
- Code Convention. Договариваемся о стиле написания.
    - *Частая интеграция.* Разработчики, по-возможности, должны интегрировать и выпускать свой код каждые несколько часов. Каждый должен работать с самой последней версией.

7. 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.

8. Уровни тестирования программного обеспечения, Верификация и валидация

Уровни тестирования

- Компонентное тестирование
- Интеграционное тестирование
- Системное тестирование
- Приемочное тестирование

Компонентное тестирование (также известное как модульное) занимается поиском дефектов и верификацией функционирования программных модулей, программ, объектов, классов и т.п., которые можно протестировать изолированно. Это может быть сделано изолированно от остальной части системы, в зависимости от контекста ЖЦ разработки и системы. В процессе могут быть использованы заглушки, драйвера и эмуляторы.

Обычно, компонентное тестирование производится с доступом к тестируемому коду и с поддержкой рабочего окружения, такого как фреймворк модульного тестирования или утилиты отладки. На практике компонентное тестирование обычно производится разработчиками, которые пишут код. Дефекты обычно исправляются сразу после того, как становятся известны, без занесения их в базу дефектов.

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 тестирование 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. Однако, это может рассматриваться как риск. Бизнес-процессы могут включать последовательность систем; могут быть важны кроссплатформенные различия.

Системное тестирование сконцентрировано на поведении тестового объекта как целостной системы или продукта. Область тестирования должна быть четко определена в главном плане тестирования либо плане тестирования для конкретного уровня тестирования.

Системное тестирование может включать тесты, основанные на рисках или спецификациях требований, бизнес-процессах, сценариях использование системы, или других высокоуровневых текстовых описаниях или моделях поведения системы, взаимодействия с ОС и системными ресурсами.

Системное тестирование должно заниматься исследованием функциональных и нефункциональных требований к системе и качеством обрабатываемых данных.

Приемочным тестированием системы чаще всего занимаются заказчики или

пользователи системы, а также другие заинтересованные лица.

Основная цель приемочного тестирования – проверка работоспособности системы, частей системы или отдельных нефункциональных характеристик системы. Поиск дефектов не является главной целью приемочного тестирования.

Приемочное тестирование оценивает готовность системы к развертыванию и использованию, хотя это не обязательно самый последний уровень тестирования. Например, крупномасштабные тесты по системной интеграции можно провести именно во время приемочного тестирования системы.

Верификация и валидация

Верификация	**Валидация**
• Компонентное тестирование • Интеграционное тестирование • Системное тестирование	• Приемочное тестирование

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)

Чтобы систематизировать исследовательское тестирование можно использовать идею туров. Туры – это идеи и инструкции по исследованию программного продукта, объединенные определённой общей темой или целью. Туры, как правило, ограничены по времени – длительность тестовой сессии не должна превышать 4 часа.

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 Виттакер использует метафору, что 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 бизнес-центру (Tours of the Business District)

Ассоциация с районом в туризме: Это район, где жители города “делают деньги”, выполняют работу. Это район, в котором туристам, часто неинтересно. Он заполнен банками, офисными зданиями. По утрам и вечерам там можно попасть в многочасовую пробку.

Сравнение с тестированием приложения: фичи, которые “делают бизнес” – они появляются в промо-материалах, ради них пользователь приобретает приложение.

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

Чтобы систематизировать исследовательское тестирование можно использовать идею туров. **Туры** – это идеи и инструкции по исследованию программного продукта, объединенные определённой общей темой или целью. Туры, как правило, ограничены по времени – длительность тестовой сессии не должна превышать 4 часа.

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 Виттакер использует метафору, что 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 историческому райнону

Ассоциация в туризме: фаворитами туристов являются музеи, отображающие античные, средневековые времена.

Сравнение с тестированием приложения: код тоже может быть “античным”. Это тот код, который давно не изменялся. Такой код, попадая в новую среду, может вообще не работать, или, подвергаясь внешнее ревизии, может оказаться непригодным. Тестировщики могут найти такой код по метке даты изменения в репозитории.

Например, если создают для iOS новый, но схожий с другим проект, то могут скопировать лишний код или ресурсные файлы, которые при проверке Apple ревьюверами может привести к отклонению приложения.

Типичные баги:

- креши
- функциональные ошибки
- несоответствие стандартам или гайдлайнам
- увеличение размера приложения

Туры по району развлечений (Tours Through the Entertainment District)

Ассоциация с районом в туризме: туристам нужны места, где можно отдохнуть, оторваться от плотного графика переездов и осмотра города.

Сравнение с тестированием приложения: в большинстве приложений есть места, где можно отвлечься от основной бизнес-задачи (сценария использования). Например, настроить приложение под свои особенные нужды, или навести “красоту” изменяя шрифт и цвет текста и т.п.

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

Чтобы систематизировать исследовательское тестирование можно использовать идею туров. **Туры** – это идеи и инструкции по исследованию программного продукта, объединенные определённой общей темой или целью. Туры, как правило, ограничены по времени – длительность тестовой сессии не должна превышать 4 часа.

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 Виттакер использует метафору, что 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 туристическому району (Tours Through the Tourist District)

Ассоциация с районом в туризме: в каждом городе есть места с наибольшей концентрацией туристов. Эти места заполнены сувенирными магазинами, ресторанами и т.п.

Сравнение с тестированием приложения: быстрые проверки, с целью просто пробежаться по функциям. Мол, “я здесь был”.

Туры по району отелей (Tours Through the Hotel District)

Ассоциация с районом в туризме: для туристов отель – это убежище от шума и суеты горящего отпуска.

Сравнение с тестированием приложения: место, где можно отвлечься от основной функциональности и популярных фич, и проверить что-то второстепенное.

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

Чтобы систематизировать исследовательское тестирование можно использовать идею туров. **Туры** – это идеи и инструкции по исследованию программного продукта, объединенные определённой общей темой или целью. Туры, как правило, ограничены по времени – длительность тестовой сессии не должна превышать 4 часа.

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 Виттакер использует метафору, что 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.

Тур по нерекомендуемым местам (The Bad-Neighborhood Tour)

Ассоциация в туризме: в каждой местности есть места, которые туристам советуют избегать.

Сравнение с тестированием приложения: в приложении – это фичи или места в коде с наибольшим скоплением багов. Про эти места вы можете знать интуитивно, а можете воспользоваться багтрекером. Как только вы нашли один баг, знайте, что, скорее всего, рядом есть и другой; он может находиться в том же функционале, а может и в соседнем. Задача тестировщика состоит в том, чтобы пройти по местам. Связано это с тем, что баги имеют свойство скапливаться в одном месте.

Тур также применим и после исправления багов. Пройдитесь взаимосвязанным областям.

Типичные баги: функциональные.

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 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.

Тестирование безопасности включает следующие шесть мер для обеспечения защищенной среды:

- Конфиденциальность. Он защищает от раскрытия информации непреднамеренным получателям.
- Целостность - позволяет передавать точную и правильную требуемую информацию от отправителей к предназначенным приемникам.
- Аутентификация - проверяет и подтверждает личность пользователя.
- Авторизация - указывает права доступа к пользователям и ресурсам.
- Доступность - это обеспечивает готовность информации о требованиях.
- Неотказание - оно гарантирует, что от отправителя или получателя не будет отказано в отправке или получении сообщения.

Тест проницаемости - рабочий процесс

Тест на проникновение включает четыре основные фазы:

1. Сбор информации (footprinting)
2. Сканирование
3. Перечисление
4. Эксплуатация

Footprinting

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.

Footprinting - Шаги

- Сбор информации
- Определение диапазона сети
- Идентификация активных машин
- Идентификация открытых портов и точек доступа
- Отпечатки пальцев OS
- Услуги по отпечатку пальцев
- Сопоставление сети

Инструменты, используемые при слежении

Ниже приведен общий набор инструментов, используемых для footprinting:

- Кто
- SmartWhois
- NSlookup
- Сэм Спейд

Другие методы, используемые при слежении

Footprinting может также включать сбор информации, такой как:

- Контактные лица компаний, адреса электронной почты и номера телефонов
- Сделки с компанией и другие вовлеченные стороны
- Новости о слияниях и поглощениях
- Ссылки на другие сайты, связанные с компанией
- Политика конфиденциальности компании

Сканирование

Сканирование - это второй шаг, который выполняется после footprinting. 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.

Инструменты, используемые в сканировании

Ниже приведен общий набор инструментов / ресурсов, используемых в Scanning:

- NMap
- Пинг
- Трассировка
- SUPERSCAN
- Netcat
- NeoTrace

Перечисление

Перечисление - следующий шаг после сканирования. 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.

Методы, используемые в перечислении

Ниже приведен общий набор процедур, используемых в перечислении:

- Идентификация уязвимых учетных записей пользователей
- Получение информации в Active Directory
- Использование snmputil для перечисления протокола Simple Network Management
- Использование DNS-запросов Windows
- Создание нулевых сеансов и соединений

Эксплуатация

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.

25. Тестирование безопасности (Виды атак)

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.

Типы эксплуатации подразделяются на три категории:

- Атака против WEB-СЕРВЕРОВ
  - SQL-инъекция
  - Межсайтовый скриптинг
  - Ввод кода
  - Захват сеанса
  - Обход каталога
- Атака на NETWORKS
  - Человек в средней атаке
  - Подделка
  - Обход брандмауэра
  - WLAN
  - Отравление ARP
- Атака на услуги
  - Переполнение буфера
  - Форматировать строки
  - ДОС
  - Ошибки аутентификации

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.

Footprinting - Шаги

- Сбор информации
- Определение диапазона сети
- Идентификация активных машин
- Идентификация открытых портов и точек доступа
- Отпечатки пальцев OS
- Услуги по отпечатку пальцев
- Сопоставление сети

Инструменты, используемые при слежении

Ниже приведен общий набор инструментов, используемых для footprinting:

Кто,SmartWhois,NSlookupСэм Спейд

Footprinting может также включать сбор информации, такой как:

- Контактные лица компаний, адреса электронной почты и номера телефонов
- Сделки с компанией и другие вовлеченные стороны
- Новости о слияниях и поглощениях
- Ссылки на другие сайты, связанные с компанией
- Политика конфиденциальности компании

Сканирование - это второй шаг, который выполняется после footprinting. 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.

Инструменты, используемые в сканировании

Ниже приведен общий набор инструментов / ресурсов, используемых в Scanning:

- NMap
- Пинг
- Трассировка
- SUPERSCAN
- Netcat
- NeoTrace

Перечисление - следующий шаг после сканирования. 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.

Методы, используемые в перечислении

Ниже приведен общий набор процедур, используемых в перечислении:

- Идентификация уязвимых учетных записей пользователей
- Получение информации в Active Directory
- Использование snmputil для перечисления протокола Simple Network Management
- Использование DNS-запросов Windows
- Создание нулевых сеансов и соединений

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.

Методы, используемые при эксплуатации

Типы эксплуатации подразделяются на три категории:

- Атака против WEB-СЕРВЕРОВ
  - SQL-инъекция
  - Межсайтовый скриптинг
  - Ввод кода
  - Захват сеанса
  - Обход каталога
- Атака на NETWORKS
  - Человек в средней атаке
  - Подделка
  - Обход брандмауэра
  - WLAN
  - Отравление ARP
- Атака на услуги
  - Переполнение буфера
  - Форматировать строки
  - Ошибки аутентификации

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.

Виды тестирования, для которых пишут чеклисты:

•Тестирование совместимости;

•Тестирование инсталляции продукта;

•Исследовательское тестирование;

•CRUD тестирование

•Тестирование на прерывания и т.д.

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.

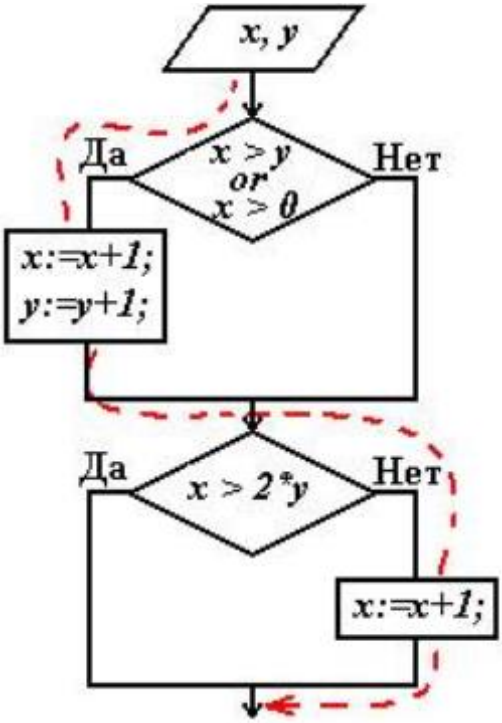
Не обязательно составлять попарные комбинации вручную, для этого существует масса инструментов.

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 (Summary, Title)** – уникальный среди tc, сост только из существительных
- **Цель тестового случая (Goal, Aim, Description)** – 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)** – есть всегда:
  * tc не пройден (not run)
  * успешно пройден (passed): ожид рез = актуальный рез
  * неудачно пройден (failed): ожид рез != актуальный рез
  * заблокирован (blocked): не может быть вып шаг либо предусловие tc
- **Уровень (Level)** – выставл в соотв с уровнем тестир, для к-го он предназн: (компонентный, интеграционный, системный, приемочный)
- **Приоритет (Priority)**:
  * высокий: д пройти в 1 очередь
  * средний: 2 очередь
  * низкий: 3 очередь
- **Автор (Author)** – в с-мах tc management выставляется авто-
- **Комментарии (Notes, Comments)**

Жизненный Цикл:

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)*

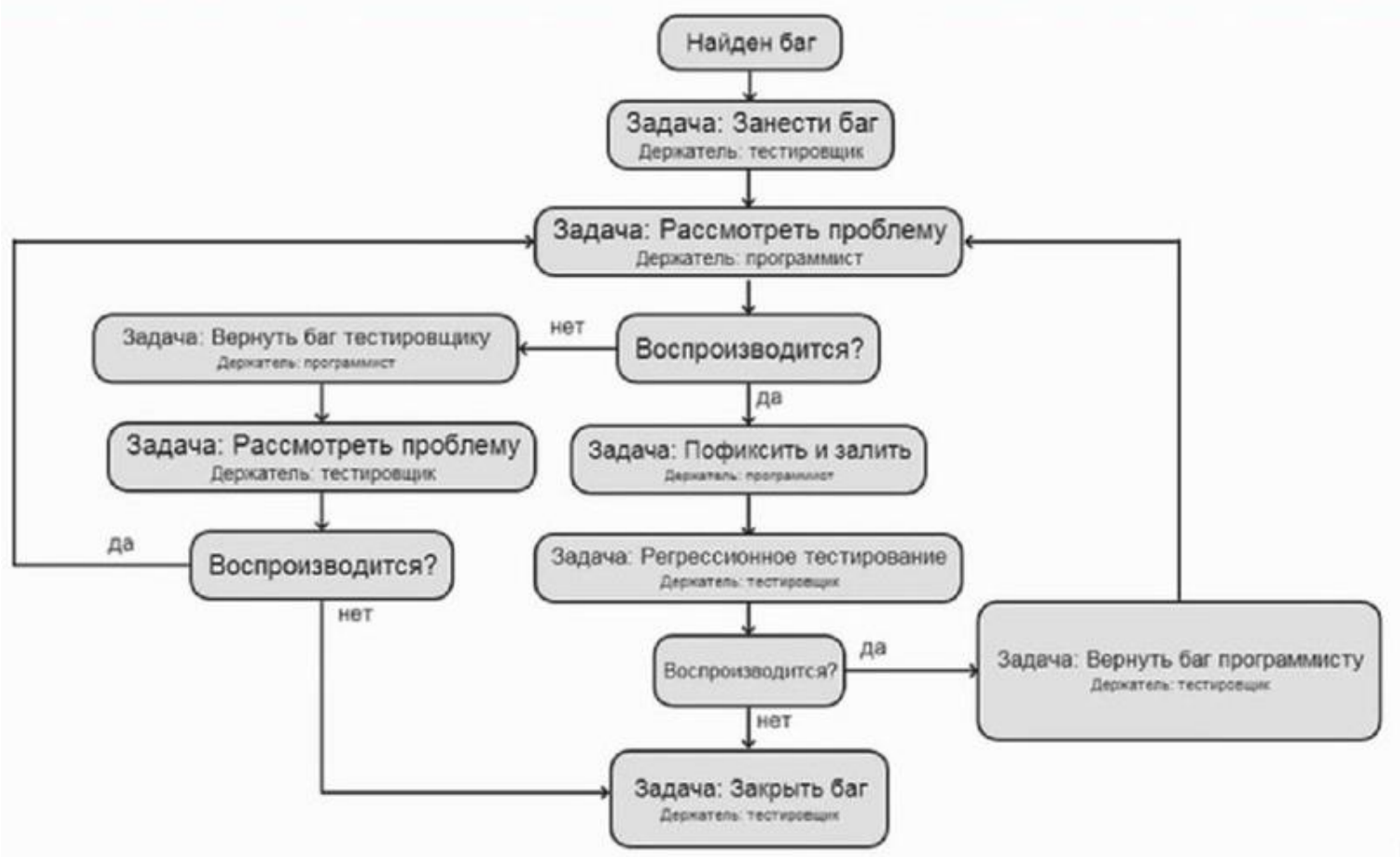
Цель отчета об ошибках:

✓**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:



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 (Интегрционные 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)
- keyword-driven подход (ключевые слова)
- behavior-driven подход (поведение)

Data-driven подход

Data-driven подход используется в приложениях, внешний вид и работа которых во многом зависит от введенных данных (калькулятор)

Keyword-driven подход

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

В подходе, который зависит от поведения системы активно используются ключевые слова, которые объединяются в глоссарии, где прописывается что выполняется в тесте, какое действие выполняется в тесте под каждым словом.

Тесты всегда описываются по структуре, используя слова

* *given* (дано),
* *when* (что делаем),
* *then*(результат (что получаем после проведенных действий))

Данный подход распространяется не только на автоматизацию, но и на весь проект в целом. Все требования оформляются в виде подобных сценариев (given, when, then).

Мануальные тестировщики на таком проекте полностью отсутствуют и все тестирование проводится посредством автоматизации.