ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Теоретические вопросы

1) Дать определение понятию тестирование. Указать основную цель

тестирования.

Тестирование программного обеспечения – проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным образом.

Цели - Распознать дефекты в объекте тестирования и увеличить вероятность того, что он при любых обстоятельствах будет работать надлежащим образом в соответствии с установленными требованиями.

2) Дать определение понятию отладка. Перечислить основные этапы в истории развития тестирования.

Отладка – процесс поиска, локализации и исправления ошибок в программе.

**Начало – 9 сентября 1947 (1945) года.**

Ученый нашли мотылька, застрявшего между контактами электромеханического реле и был добавлен в дневник с сопроводительной записью.

**В 60-х годах** прошлого века основное внимание уделялось т.н. «исчерпывающему тестированию» - проверке всех возможных путей выполнения кода со всеми возможными входными данными.

**В 70-х годах** прошлого столетия фактически родились две фундаментальные идеи тестирования: тестирование сначала рассматривалось как процесс доказательства работоспособности программы в некоторых заданных условиях (positive testing), а затем — строго наоборот: как процесс доказательства неработоспособности программы в некоторых заданных условиях (negative testing).

**Во второй половине 70-х** тестирование представляло собой выполнение программы с целью найти ошибки, а не доказать, что она работает. Успешный тест – такой, который обнаруживает ранее неизвестные проблемы.

**С середины 80-х** тестировщики стали вовлекаться во все аспекты жизненного цикла разработки (software lifecycle).

**В 90-х годах** произошёл переход от тестирования как такового к более всеобъемлющему процессу, который называется «обеспечение качества» (quality assurance), охватывает весь цикл разработки ПО.

**В 2000-е** развитие тестирования продолжалось в контексте поиска всё новых и новых путей, методологий, техник и подходов к обеспечению качества.

3) Перечислить объекты тестирования. Описать основную проблему и задачи тестирования.

Объекты –

1. Программы при их непосредственном запуске и исполнении
2. Код программ без запуска и исполнения
3. Прототип программного продукта
4. Проектная документация
5. Сопроводительная документация

Основная проблема тестирования – определение достаточности множества тестов для истинности вывода о правильности реализации программы, а также нахождения множества тестов, обладающего этим свойством.

Проверить корректную работу всех функциональных требований заказчика к продукту.

1. Обеспечить надежность и безопасность продукта на предмет постороннего вмешательства или неправомерного доступа к конфиденциальной информации.
2. Оценить оптимальную нагрузку, а также способность сохранять работоспособность в период критической нагрузки.
3. Убедиться, что продукт «интуитивно» удобен и не содержит непонятных функций, которые могут вводить в заблуждение пользователей.
4. Проверить, что продукт не только исправно работает как отдельно взятая система, но и не влияет на работу других программ.
5. **Повысить качество программного продукта**
6. **Снизить стоимость разработки путем раннего обнаружения дефектов**

4) Дать описание процессов Quality Assurance и Quality Control.

Перечислить основные задачи этих процессов.

**Обеспечение качества** (QA, Quality Assurance) – Совокупность планируемых и систематически осуществляемых процессов, процедур, операций и отдельных мероприятий, необходимых для создания уверенности в том, что продукция удовлетворяет определенным требованиям к качеству.

**Контроль качества** (QC, Quality Control) – Тестирование + принятие решения о выпуске продукта (следующий уровень - контроль качества продукта - анализ результатов тестирования и качества «билдов», в процессе разработки).

5) Перечислить основные задачи тестировщика. Описать основные

принципы тестирования.

**Принцип 1 – Тестирование демонстрирует наличие дефектов**

Тестирование может обнаружить дефекты в программе, но не может доказать их полное отсутствие. Важно разработать тест-кейсы, которые максимально возможным образом выявят ошибки. С достаточным охватом тестирования вероятность наличия дефектов в программном обеспечении может быть снижена. Однако, даже если дефекты не были обнаружены в процессе тестирования, нельзя утверждать, что их точно нет.

**Принцип 2 – Исчерпывающее тестирование недостижимо**

Полное тестирование всех возможных комбинаций вводов и предусловий невозможно, за исключением тривиальных случаев. Поэтому вместо этого следует использовать анализ рисков и установку приоритетов, чтобы сосредоточить усилия наиболее эффективно.

**Принцип 3 – Раннее тестирование**

Для раннего обнаружения дефектов тестирование должно начинаться как можно раньше в жизненном цикле разработки программного обеспечения или системы и быть сфокусированным на конкретных целях.

**Принцип 4 – Скопление дефектов**

Усилия по тестированию должны быть распределены пропорционально фактической плотности дефектов в различных модулях системы. Основная часть критических дефектов обычно находится в ограниченном числе модулей, что соответствует принципу Парето: 80% проблем обусловлены 20% модулей.

**Принцип 5 – Парадокс пестицида**

Повторное выполнение одних и тех же тестов множество раз может привести к утрате эффективности, поскольку новые дефекты могут перестать быть обнаруживаемыми. Чтобы предотвратить это, важно создавать разнообразные тесты, которые охватывают все компоненты программного обеспечения или системы, чтобы максимально обнаружить возможные дефекты.

**Принцип 6 – Тестирование зависит от контекста**

Тестирование различается в зависимости от контекста. Критически важное программное обеспечение, где безопасность играет ключевую роль, требует особого подхода к тестированию, отличного от тестирования электронной коммерции. Программное обеспечение для медицинских целей требует более строгой и тщательной проверки, чем компьютерные игры. Кроме того, сайты с высокой посещаемостью должны пройти тестирование производительности, чтобы убедиться в их способности работать при высоких нагрузках.

**Принцип 7 – Заблуждение об отсутствии ошибок**

Тот факт, что тестирование не обнаружило дефектов, еще не значит,

что программа готова к релизу. Нахождение и исправление дефектов

будут не важны, если система окажется неудобной в использовании,

и не будет удовлетворять ожиданиям и потребностям пользователя.

6) Перечислить основные направления тестирования. Описать основные задачи направлений.

Направления тестирования:

1. Статическое (без запуска программного кода продукта)

Раннее выявление дефектов (дешевле)

Проверяется:

* 1. код
  2. требования
  3. спецификации
  4. архитектура
  5. дизайн

1. Динамическое (при запущенной системе)

Более позднее выявление / более дорогое устранение дефектов (спецификации написаны, архитектура разработана, интерфейс программы тоже готов)

Проверяется:

1. все характеристики качества запущенного приложения

7) Перечислить методы тестирования. Описать каждый метод, указать кто проводит каждый метод тестирования.

1. Метод белого ящика (white box testing) - смотрим код

Тестирование "белый ящик" выполняется с целью обнаружения проблем во внутренней структуре программы.

Объектами тестирования в этом случае являются данные, полученные путем анализа логики программы.

Общая задача такого тестирования - обеспечить проверку каждого шага по алгоритму программы.

Обычно тестирование "белый ящик" проводится разработчиками.

1. Метод черного ящика (black box testing) - не заглядываем внутрь системы

Под «чёрным ящиком» понимается объект исследования, внутреннее устройство которого неизвестно, т.е. как именно работает программа считается несущественным.

Этот подход до сих пор является самым распространенным.

Объектами тестирования в этом случае являются потоки входных и выходных данных. Это позволяет определять «правильность» работы ПО в соответствии с функциональными требованиями к продукту.

1. Метод серого ящика (gray box testing) – частично заглядываем в систему

В исследованиях по методу "серого ящика» объединены методы "белого ящика" и способы тестирования с помощью входных данных по методу "черного ящика".

При работе этим методом подразумевается, что тестировщик имеет доступ к внутреннему устройству программы, но тестирование производит с точки зрения конечного пользователя.

Удачным примером простого анализа по методу "серого ящика" является запуск программы внутри отладчика и подача на вход этой программы различных данных.

8) Дать понятие определению тестовое покрытие. Описать уровни по глубине тестирования.

Тестовое покрытие - Это одна из метрик оценки качества тестирования, представляющая из себя плотность покрытия тестами требований либо исполняемого кода

1. Дымовое (smoke test)

Направлено на проверку самой главной, самой важной, самой ключевой функциональности, неработоспособность которой делает бессмысленной дальнейшее тестирование. Самый первый и быстрый уровень. Если тесты этого уровня не проходят, то тестирование прекращается.

**Пример**: запускается ли программа, устанавливается или удаляется ли она вообще.

1. Критического пути (critical path test)

Проверка функциональности, используемой типичными пользователями в типичной повседневной деятельности. В рамках данного тестирования, как правило, проверяется большинство требований, предъявляемых к программному продукту.

Т.е. проверяем правильность работы часто используемых функций, **например**: возможность войти на сайт выбрать аккаунты для публикации, создать и опубликовать пост.

1. Расширенное (extended test)

Проверка всей (остальной) функциональности, заявленной в требованиях. Это углубленный тест, при котором проверяется нестандартное использование программного продукта. Прогоняются сложные, логически запутанные сценарии, совершаются действия, которые конечный пользователь будет совершать редко.

**Пример**: границы переполнения массивов, ввод специальных символов

9) Описать уровни тестирования по уровню детализации приложения

«вширь»). Назвать классификацию по привлечению конечных пользователей.

1. **Компонентное/модульное**(component / unit testing)
2. **Интеграционное** (integration testing)
3. **Системное** (system testing)
4. **Приёмочное** (Acceptance testing)

Классификацию по привлечению конечных пользователей

1. **Альфа-тестирование** (alpha testing) **выполняется внутри организации-**

**разработчика** с возможным частичным привлечением конечных

пользователей. Может являться формой внутреннего приёмочного

тестирования. Суть этого вида: продукт уже можно периодически

показывать внешним пользователям, но он ещё достаточно «сырой»,

потому основное тестирование выполняется организацией-

разработчиком.

1. **Бета-тестирование** (beta testing) выполняется **вне организации-**

**разработчика с активным привлечением конечных пользователей**/

заказчиков. Может являться формой внешнего приёмочного

тестирования. Суть этого вида: продукт уже можно открыто показывать

внешним пользователям, он уже достаточно стабилен, но проблемы всё

ещё могут быть, и для их выявления нужна обратная связь от реальных

пользователей.

1. **Гамма-тестирование** (gamma testing) — **финальная стадия** тестирования

перед выпуском продукта, направленная на исправление

незначительных дефектов, обнаруженных в бета-тестировании. Как

правило, также выполняется с максимальным привлечением конечных

пользователей/заказчиков. Может являться формой внешнего

приёмочного тестирования. Суть этого вида: продукт уже почти готов, и

сейчас обратная связь от реальных пользователей используется для

устранения последних недоработок

10) Перечислить виды тестирования. Описать цели функционального

тестирования.

1. **Функциональные виды**
2. **Нефункциональные виды**
3. **Виды, связанные с изменениями**

Цели функционального тестирования:

1. тестирование каждой функции в отдельности;
2. обнаружение дефекта в программном продукте;
3. определение степени соответствия программного продукта требованиям и ожиданиям заказчика;
4. принять решение о возможности передачи продукта заказчику.

11) Перечислить нефункциональные виды тестирования. Описать

несколько нефункциональных видов (не менее пяти).

1. Тестирование производительности (performance testing)

Тестирование, которое проводится с целью определения, как быстро работает система или её часть под определённой нагрузкой.

В рамках тестирования производительностивыделяют следующие подвиды:

a)нагрузочное (performance & load testing)

b)стрессовое ( stress testing)

c)тестирование стабильности / надежности

(stability / reliability testing)

d)тестирование объемами (volume testing)

1. Тестирование установки (installation testing)
2. Тестирование удобства пользования (usability testing)
3. Тестирование на отказ и восстановление (failover & recovery testing)
4. Конфигурационное тестирование (configuration testing)
5. Тестирование интернационализации (internationalization testing)
6. Локализационное тестирование (localization testing)
7. Тестирование документации (document testing)
8. Тестирование совместимости (compatibility testing)

12) Перечислить виды тестирования, связанные с изменениями. Описать основной вид.

1. **Дымовое тестирование** (Smoke Testing)

Дымовое тестирование рассматривается как короткий цикл тестов, выполняемый для подтверждения того, что после сборки кода (нового или исправленного) устанавливаемое приложение, стартует и выполняет основные функции. Для облегчения работы, экономии времени и людских ресурсов рекомендуется внедрять автоматизацию тестовых сценариев для дымового тестирования.

1. **Регрессионное тестирование** (RegressionTesting)

Это вид тестирования, осуществляемый для подтверждения факта, что существующая ранее функциональность работает как и прежде после сделанных в приложении или окружающей среде исправлений или дополнений (устранение дефекта, слияние кода, миграция на другую операционную систему, базу данных, веб-сервер или сервер приложения).

1. **Тестирование сборки** (Build Verification Test)

Тестирование, направленное на определение соответствия, выпущенной версии, критериям качества для начала тестирования. По своим целям является аналогом дымового тестирования, направленного на приемку новой версии в дальнейшее тестирование или эксплуатацию. Вглубь оно может проникать дальше, в зависимости от требований к качеству выпущенной версии.

1. **Санитарное тестирование** или проверка согласованности/исправности (Sanity Testing)

Это узконаправленное тестирование достаточное для доказательства того, что конкретная функция работает согласно заявленным в спецификации требованиям. Является подмножеством регрессионного тестирования. Используется для определения работоспособности определенной части приложения после изменений произведенных в ней или окружающей среде. Обычно выполняется вручную.

13) Дать определение понятию качество ПО. Описать, что представляет

собой качественное ПО.

Качество программного обеспечения — это степень, в которой ПО обладает требуемой комбинацией свойств. ИЛИ Качество программного обеспечения – это совокупность характеристик ПО, относящихся к его способности удовлетворять установленные и предполагаемые потребности.

Качественный – значит соответствующий ожиданиям того, кому этот продукт предназначается.

14) Перечислить основные модели качества ПО. Описать текущую

стандартную модель качества ПО.

Наиболее известные модели качества:

➢ **модель МакКола и др.**

Предложенная в 1977 г. Дж. МакКолом (J.McCall), П. Ричардсом (P. Richards) и Дж. Уолтерсом (G. Walters) модель качества ПО подразделяет атрибуты качества на три категории:

✓ Факторы (factors) – описывают ПО с точки зрения пользователя, определяются требованиями и группируются по видам деятельности заинтересованных лиц

✓ Критерии (criteria) – числовые уровни факторов: описывают ПО с точки зрения разработчика, задаются как цели при разработке

✓ Метрики (metrics) – служат для количественного описания и измерения качества (оценки от 0 до 10)

➢ **модель Боэма**

В 1978 Боэм предложил свою модель, по существу ьпредставляющую собой расширение модели МакКола. Атрибуты качества группируются по способуь использования ПО (primary use). В дополнение к факторам МакКола атрибуты качества по Боэму включают следующие:

✓ ясность (clarity)

✓ удобство внесения изменений (modifiability)

✓ документированность (documentation)

✓ способность к восстановлению функций (resilience)

✓ понятность (understandability)

✓ адекватность (validity)

✓ функциональность (functionality)

✓ универсальность (generality)

✓ экономическая эффективность (economy)

➢ **модель ISO 9126/ ISO 25000**

В 1991 году в качестве стандартной была принята модель качества ПО ISO 9126. Эта модель не является прямым расширением ранее предложенных. В ней оценка качества ПО основана на трехуровневом рассмотрении.

 Цели (goals) – то, что мы ходим видеть в ПО

 Атрибуты (attributes) – свойства ПО, показывающие приближение к цели

Метрики (metrics) – количественные характеристики степени наличия атрибутов

15) Описать цели и атрибуты качества ISO 9126. Привести примеры

оценки качества ПО.

16) Дать определение понятию валидация. Описать основные методы

контроля качества.

Валидация - подтверждение того, что система соответствует ожиданиям заказчика при ее непосредственном применении. / проверка продукта на соответствие потребностям пользователя

17) Дать определение понятию верификация. Описать основные задачи

верификации.

Верификация – проверка продукта на соответствие входным данным (~ проектным требованиям, спецификациям) / проверка того, что система соответствует установленным требованиям

Валидация отвечает на вопрос

❖ «Сделано ли то, что нужно?»

❖ «Соответствует ли разработанная система ожиданиям заказчика?».

18) Перечислить методы верификации. Описать несколько методов

верификации (не менее трех).

**Статические методы** верификации выявляют неверные конструкции или неверные отношения объектов программы (ошибки формального задания) формальными методами анализа без выполнения тестируемой программы:

✓ С помощью специальных инструментов контроля кода

✓ Обзоры (Reviews)

✓ Инспекции (Inspections)

✓ Сквозные просмотры (Walkthroughs)

✓ Аудиты (Audits)

✓ Тестирование требований, спецификаций, документации.

**Динамические методы** верификации осуществляют выявление ошибок на выполняющейся программе. Верификация заканчивается, когда выполнилось или "прошло" (pass) успешно достаточное количество тестов в соответствии с выбранным критерием тестирования

19) Дать определение понятию требование. Описать важность требований.

**Требования** ***(requirements)*** – описание того, какие функции и с соблюдением каких условий должно выполнять приложение в процессе решения полезной для пользователя задачи.

**Требования:**

1. Позволяют понять, что и с соблюдением каких условий система должна делать.
2. Предоставляют возможность оценить масштаб изменений и управлять изменениями.
3. Являются основой для формирования плана проекта.
4. Помогают предотвращать или разрешать конфликтные ситуации.
5. Упрощают расстановку приоритетов в наборе задач.
6. Позволяют объективно оценить степень прогресса в разработке проекта.

Требования являются отправной точкой для определения того, что проектная команда будет проектировать, реализовывать и тестировать.

Вне зависимости от того, какая модель разработки ПО используется на проекте, чем позже будет обнаружена проблема, тем сложнее и дороже будет её решение.

Если проблема в требованиях будет выяснена на стадии планирования, её решение может свестись к исправлению пары слов в тексте, в то время как недоработка, вызванная пропущенной проблемой в требованиях и обнаруженная на стадии эксплуатации, может даже полностью уничтожить проект.

20) Перечислить типы и уровни требований. Описать объекты требований.



Чаще всего выделяют два типа требований:

* **Функциональные**
* **Нефункциональные**

**Функциональные требования** определяют «***Что система******должна делать?****»*

**Нефункциональные требования** определяют «***С соблюдением каких условий должна делать?***»

21) Перечислить уровни требований. Описать группу функциональных требований.

**Бизнес-требования** *(business requirements)* – определяют высокоуровневые цели организации или клиента (потребителя) – заказчика разрабатываемого программного обеспечения.

**Пользовательские** **требования** *(user requirements)* – описывают цели/задачи пользователей системы, которые должны достигаться/выполняться пользователями при помощи создаваемой программной системы.

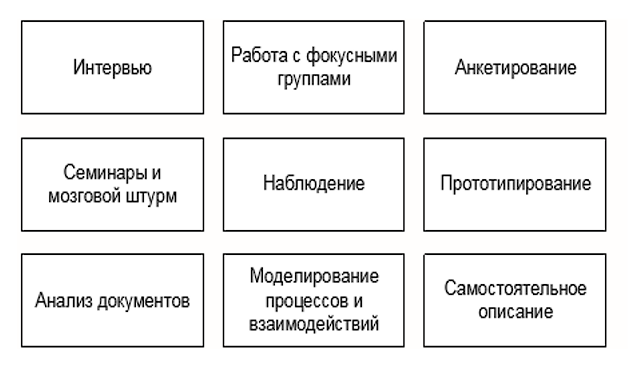
**Функциональные требования** *(functional requirements)* – определяют функциональность (поведение) программной системы, которая должна быть создана разработчиками для предоставления возможности выполнения пользователями своих обязанностей в рамках бизнес-требований и в контексте пользовательских требований.

22) Описать нефункциональные требования. Перечислить пути выявления требований.

Описывают *"каким должно быть ПС , в каких условиях должно работать, как хорошо она это делает"*

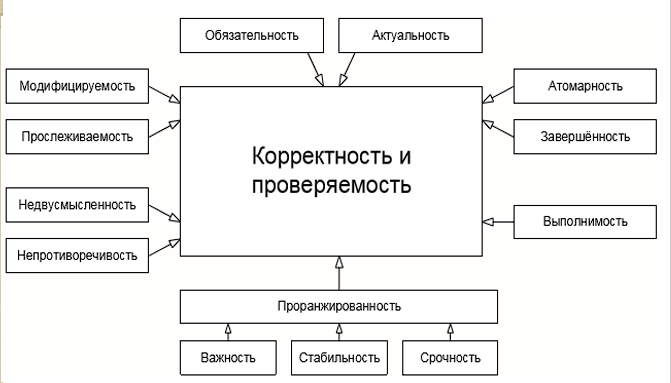
1. Внешние интерфейсы
   1. Интерфейс пользователя,
   2. Аппаратные интерфейсы,
   3. Программные интерфейсы,
   4. Коммуникационные интерфейсы
2. Атрибуты качества
   1. Удобство использования
   2. Надежность
   3. Производительность
   4. Эксплуатационная пригодность (способность к сопровождению)
3. Системные ограничения
   1. Требования, выдвигаемые ИС к среде своего функционирования (тактовая частота процессора, объем памяти, требования к выбору операционной системы)

Пути выявления требований:



23) Описать характеристики качественных требований. Перечислить

основные характеристики набора требований.



* **Полнота (отдельного требования и системы требований)** — требование должно содержать всю необходимую информацию для его реализации. В него включается вся информация об описываемом параметре, известная на момент описания. Система требований также не должна содержать не выявленных и не определенных требований. Причины неполноты описания следует явно объявлять.
* **Корректность отдельного требования и согласованность (непротиворечивость) системы требований** — требование не должно содержать в себе неверной, неточной информации, а отдельные требования в системе требований не должны противоречить друг другу.
* **Однозначность (ясность, непротиворечивость)** — требование должно быть внутренне непротиворечиво и все работающие с ним должны понимать его одинаково. Требования следует выражать просто, кратко и точно, используя известные термины.
* **Осуществимость (выполнимость) —** включаемое в спецификацию требование должно быть выполнимым при заданных ограничениях операционной среды. В частности, для нефункциональных требований проверяется возможность достижения указанных численных значений при существующих ограничениях.
* **Необходимость —** требование должно отражать возможность или характеристику ПО, действительно необходимую пользователям, или вытекающую из других требований.
* **Проверяемость (верифицируемость)** — проверяемость требования означает, что существует конечный и разумный по стоимости процесс ручной или машинной проверки того, что ПО удовлетворяет этому требованию.
* **Доступность** — атрибут качества, определяющий время непрерывной работы приложения или системы. Чтобы определить этот параметр, обычно указывают максимально допустимое время простоя системы.
* **Надежность** — требование, описывающее поведение приложения или системы в нештатных ситуациях (примеры: автоматический перезапуск, восстановление работы, сохранение данных, дублирование важных данных, резервирование логики)

24) Описать важность тестирования требований. Перечислить проблемы

с требованиями.

Важность тестирования требований состоит в том, что хорошие требования позволяют:

* Достичь **общего пони**мания между заказчиком и разработчиком
* Определить **рамки проекта**
* Более точно определить **финансовые и временные характеристики** проекта.
* **Обезопасить заказчика** от риска получить продукт, в котором он не сможет работать
* **Обезопасить разработчика** от риска попасть в ситуацию «неконтролируемого размытия границ», которое может привести к непредвиденным затратам ресурсов сверх начальных ожиданий.

Проблемы

* Проблема незавершенности (неполноты)
* Проблема двусмысленности
* Проблема противоречивости
* Проблема некорректности
* Проблема непроверяемости
* Проблема немодифицируемости
* Проблема непрослеживаемости
* Проблема непроранжированности (неупорядоченности)

25) Дать определению артефакта тест-план. Описать основные задачи

этапа планирования.

1. Обеспечить ключевых участников проекта информацией о том, каким образом будет строиться процесс тестирования в проекте.

2. Убедиться, что тест-план реалистичен, полон и выполним.

26) Перечислить действия тестировщиков на стадии планирования. Описать сложности планирования.

✓ Понять, что за продукт будет тестироваться, как он работает и для чего он нужен

✓ Понять, как продукт будет использоваться

✓ Хорошо изучить требования к продукту

✓ Решить какие части продукта будут тестироваться и как

✓ Убедиться, что выбранные области в полном объеме покрыты тестами

✓ Решить, какие из методов и техник тестирования наиболее эффективны для проверки нашего продукта

✓ Определить критерии качества продукта

✓ Определить и приоритизировать риски, то есть ситуации, которые приведут к ухудшению качества программного продукта. А также подумать о том, как предотвратить возникновение подобных ситуаций, и как из них выйти

**Сложности планирования**

Трудно расставить приоритеты

Возникают различные ограничения

27) Описать назначение тест-плана. Перечислить основные секции тест-плана.

➢ служит для поиска ошибок

➢ облегчает управление работами и контроль хода их выполнения

➢ облегчает организацию технических аспектов тестирования

➢ помогает организовать и скоординировать усилия сотрудников, разрабатывающих и тестирующих программный продукт

➢ повышает эффективность и полноту тестирования

Секции тестового плана включают в себя:

➢Перечень работ

➢Критерии качества и оценка качества процесса

➢Оценка рисков

➢Документация и письма

➢Тестовая стратегия

➢Ресурсы

➢Метрики

➢Расписание

➢Ключевые точки

28) Описать характеристики хорошего тест-плана. Какие группы рисков

существуют?

В тест плане должны быть:

✓ определены **цели** тестирования, тестовый подход, стратегия, методы, виды тестирования. Запланированный подход должен быть реально выполним

✓ установлены реалистичные **критерии качества**.

✓ определены критерии прохождения приёмочного теста и условия прекращения тестирования.

✓ определены все **артефакты**, подлежащие сдаче, поставке или распространению (заказчику, проекту и т.д.)

✓ перечислены **тестовые ресурсы** (люди, оборудование) с указанием ролей, назначений, ответственности.

Также:

✓ Должно быть определено и описано тестовое оборудование, окружение, программное обеспечение.

✓ Должен быть определён график тестирования, он должен быть реалистичен и выполним.

✓ Тест-план должен соответствовать принятому в компании шаблону, если на проекте не решено иначе: например, использовать шаблон заказчика.

**Группы рисков**

➢**Человеческие** – недостаток квалификации, нехватка сотрудников, внутренние взаимоотношения, больничные листы (10%-20%), форс-мажор, отпуска, увольнение

➢**Программно-аппаратные** – нехватка оборудования, выход из строя, устаревшее оборудование, поздняя постановка ПО, форс-мажор, отключение электричества (Internet)

➢**Финансовые** - деньги

29) Дать определение понятию классы эквивалентности. Перечислить техники тест-дизайна.

Класс эквивалентности (equivalence class) – набор данных, обрабатываемый одинаковым образом и приводящий к одинаковому

результату.

30) Дать определение понятию граничные условия. Привести пример

использования классов эквивалентности и граничных условий.

Граничные условия (или просто – границы) – это те места, в которых один класс эквивалентности переходит в другой.

В нашем примере программа позволяет вводить максимум 2 символа и текстовое поле принимает

только числа от 0 до 99. Поэтому граничные значения для этого поля -1,0 и 99, 100

31) Дать определение артефакту чек-лист. Описать основное назначение чек-листа.

32) Дать определение понятию позитивное тестирование. Привести

пример позитивного теста.

Позитивное тестирование – это тестирование с применением сценариев, которые

соответствуют нормальному (штатному, ожидаемому) поведению системы. С его

помощью можно определить, что система делает то, для чего и была создана.

33) Дать определение понятию негативное тестирование. Привести

пример негативного теста.

Негативным называют тестирование, в рамках которого применяются сценарии, которые соответствуют внештатному поведению тестируемой системы. Это могут быть, например, исключительные ситуации или неверные данные.

34) Дать определение артефакту тест-кейс. Описать свойства тест-кейса.

35) Перечислить разделы артефакта тест-кейс. Описать основные

разделы тест-кейса (не менее пяти)

36) Дать определение понятию дефект (баг). Описать, как определяют дефекты в программе.

**Дефект** – это несоответствие требованиям или функциональным спецификациям.

37) Перечислить три условия, при которых существует дефект (баг).

Описать жизненный цикл дефекта.

**Новый** (New). Когда тестировщик находит дефект, то он представляет его на рассмотрение в систему управления дефектами. С этого момента баг начинает свою официальную жизнь и о его существовании знают необходимые люди.

**Назначен** (assigned). Далее разработчик рассматривает дефект и назначает его исправление кому-то из команды разработчиков.

**Исправлен** (fixed). Разработчик, которому было назначено исправление дефекта, исправляет его и сообщает о том, чтозадание выполнено.

**Проверен** (verified). Тестировщик, который обнаружил ошибку проверяет на новом билде (в котором исправление данной ошибки заявлено), исправлен ли дефект на самом деле. И только в том случае, если ошибка не проявится на новом билде, тестировщик меняет статус бага на Verified.

**Открыт заново** (reopened). Если баг проявляется на новом билде, тестировщик снова открывает этот дефект. Баг приобретает статус Reopened.

**Отклонён** (reject) – не баг (not a bug). Баг может быть отклонён. Во-первых, потому, что для заказчика какие-то ошибки перестают быть актуальными. Во-вторых, это может случится по вине тестировщика из- за плохого знания продукта, требований (дефекта на самом деле нет).

**Отложен** (deferred). Если исправление конкретного бага сейчас не очень важно или заказчик пока думает, или мы ждём какую-то информацию, от которой зависит исправление бага, тогда баг приобретает статус Deferred.

**Дублирован** (duplicate). Если уже существует открытый баг, который направлен на выявление той же самой ошибки.

**Не могу воспроизвести** (can not reproduce). Если в описании баг нет всей необходимой информации, программист не сможет его воспроизвести, что бы починить.

**Не чинить** (will not fix). Формально это баг, но чинить его не будут, не могут или не хотят 38) Дать определение артефакту баг-репорт. Описать цель написания отчета.

39) Перечислить атрибуты баг-репорта. Описать основные разделы (не

менее пяти)

**◦ Идентификатор (id)**

**◦ Краткое описание (summary)**

**◦ Подробное описание (description)**

**1) Шаги воспроизведения (steps to reproduce, STR)**

**2) Актуальный результат (Actual Result)**

**3) Ожидаемый результат (Expected Result)**

**◦ Воспроизводимость (reproducible)**

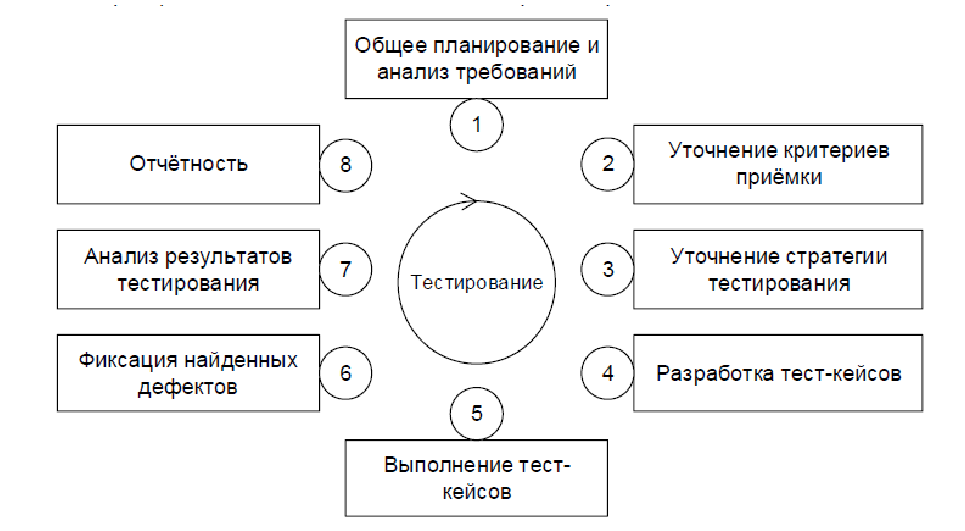
**◦ Важность (severity)**

**◦ Срочность (priority)**

**◦ Симптом (symptom)**

40) Описать жизненный цикл тестирования. Перечислить сложности

тестирования.



1. ПО по своей природе является **концептуальным**. В отличие от, например, здания или любого другого физического объекта, сложно «посмотреть» на программный продукт и оценить степень его завершённости.

2. **Недостаток контроля.** Поскольку ПО является нематериальным в физическом смысле, его сложно контролировать. Без точной оценки процесса разработки срываются графики выполнения работ и превышаются установленные бюджеты. Очень сложно оценить объём выполненной и оставшейся работы.

3. **Отсутствие связи между отдельными событиями проекта приводит к его провалу.** Ценность прослеживаемости особенно высока в ситуациях, когда в одной из версий продукта возникает проблема, которая оказывает влияние на другие версии этого продукта или на другие продукты. Отсутствие связи между событиями проекта приводит к тому, что решение одной проблемы увеличивает проблему в другой области или приводит к неудаче в попытке решить аналогичную проблему где-то в другом месте.

4. **Недостаток мониторинга**. Невозможно выполнить мониторинг проекта, если у менеджера проекта нет инструментальных средств, чтобы следить за фактической разработкой продукта в пределах проекта. В такой ситуации руководство не может принимать компетентные решения, поэтому графики продолжают срываться, а затраты продолжают превышать установленный бюджет.

5. **Неконтролируемые изменения.** ПО является достаточно гибким, оно представляет результат работы большого коллектива, но у потребителей постоянно возникают новые идеи относительно данного программного продукта. В крупных проектах рекомендуется использовать инструментальные средства, позволяющие гибко управлять изменениями в проекте.

6. **Групповой синдром разработчика.** Если для разработки проекта требуется более одного разработчика, то возникает проблема группы людей, работающих над одной «базой продукта». Усилия тратятся впустую, если несколько человек работают над одним итем же файлом, а затем его сохраняют – сохраняются только изменения, записанные тем, кто работал над файлом последним. Самое простое решение этой проблемы лежит в блокировании файла на время работы с ним одним из сотрудников, чтобы предотвратить одновременную работу над ним нескольких человек.

7. **Множественность версий.** Совершенствование базового продукта приводит к выпуску дополнительных версий с самыми последними изменениями. Если в программе обнаружены ошибки, изменения необходимо сделать во всех версиях. Как только в продукте появляются новые свойства, они должны быть доступны для всех пользователей независимо от времени выпуска версии продукта. Основные сложности тестирования

8. Семейство программных продуктов. Если программный продукт работает с четырьмя версиями Windows, тремя версиями Unix, Linux и др. платформами, то для этих платформ необходимы разные процессы установки ПО, настройки, особенности реализации и т.д.

9. **Изменение графика работ.** Поскольку технические требования меняются в процессе работы с проектом, должен меняться и график их выполнения.

10. **Изменения штата сотрудников.**

11. **Изменения во внешней среде.** Некоторые проекты разрабатываются на протяжении достаточно длительного времени (история знает проекты, разрабатываемые более 15-ти лет). За это время меняются не только требования к проекту, меняется очень многое во всей IT-индустрии. Выходят новые версии ОС, появляется новое аппаратное обеспечение, появляются новые технологии. Для успешного выполнения проекта такие изменения следует учитывать ещё на стадии планирования.

41) Описать момент завершения тестирования. Описать три любых

эвристики завершения тестирования

Программа считается готовой к выпуску, когда **устранены абсолютно все критические ошибки и ~85 % не критических ошибок.** Считается, что дальнейшее тестирование экономически не целесообразно.

1. **Эвристика «Время вышло!»**. Для многих специалистов по тестированию это наиболее распространенная эвристика: тестирование останавливается, когда заканчивается выделенное на него время.
2. **Эвристика «мертвой лошади»** (The Dead Horse Heuristic). В программе слишком много ошибок, так что продолжение тестирования не имеет смысла. Мы знаем, что все изменится настолько, что сведет на нет результаты текущего тестирования.
3. **Эвристика «освежающей паузы»** (The Pause That Refreshes Heuristic). Предполагаея приостановку тестирования на некоторое время. Можно остановить тестирование и сделать перерыв, когда устали, когда стало скучно или пропало вдохновение. Можно сделать паузу на то, чтобы выполнить некоторые исследования, разработать планы, поразмыслить над тем, что делали в прошлом и понять, что делать дальше

42) Перечислить эвристики завершения тестирования. Описать в каких

случаях применяют эвристики (не менее пяти)

1. **Эвристика «Время вышло!».**
2. **Эвристика «мертвой лошади» (The Dead Horse Heuristic).**
3. **Эвристика «освежающей паузы» (The Pause That Refreshes Heuristic).**
4. Эвристика «Отсутствие продвижения» (The Flatline Heuristic).
5. Эвристика «Больше нет интересных вопросов» (No more interesting questions).
6. Эвристика «Пиньяты» (The Piñata Heuristic).
7. Эвристика «Задание выполнено» (The Mission Accomplished Heuristic).
8. Эвристика «Отмена задания» (The Mission Accomplished Heuristic).
9. Эвристика «Зашел в тупик» (The I Feel Stuck! Heuristic)
10. Эвристика «Привычного завершения» (The Customary Conclusion Heuristic).
11. Эвристика «Уклонения/безразличия» (The Avoidance/Indifference Heuristic).

43) Дать определение артефакту отчет о результатах тестирования.

Описать основную цель и задачу отчета.

Цель написания TRR – предоставление лицам, заинтересованным в проекте, полной и объективной информации о текущем состоянии качества проекта. Эта информация выражается в конкретных фактах и цифрах.

Его основная задача – оценить текущее или финальное качество проекта и принять (если необходимо) – соответствующие решения и меры.

44) Перечислить разделы отчета о тестировании. Описать финальный

отчет о результатах тестирования.

**Разделы отчета о тестировании**

1. Команда тестировщиков

2. Описание процесса тестирования (testing

process description)

3. Краткое описание (summary)

4. Расписание (testing timetable)

5. Рекомендации (recomendations)

6. Статистика по ошибкам (bugs statistics)

7. Список новых ошибок (new bugs found)

8. Статистика по всем ошибкам (all bugsstatistics)

В конце работы с проектом формируется ещё один отчёт о результатах тестирования – финальный.

В дополнение к уже рассмотренным разделам такой отчёт включает описание и анализ существовавших на проекте проблем и найденных эффективных решений. Такой отчёт обсуждается на общем собрании проектной команды, где по результатам обсуждения формируются и документируются выводы, направленные на избежание в будущем проблем, возникших на данном проекте, а также направленные на накопление позитивного опыта с

целью применения его в будущих или выполняемых параллельно проектах.

45) Дать определение понятию модульное тестирование. Описать тема 3.2

использование структурных критериев при проведении просмотра.

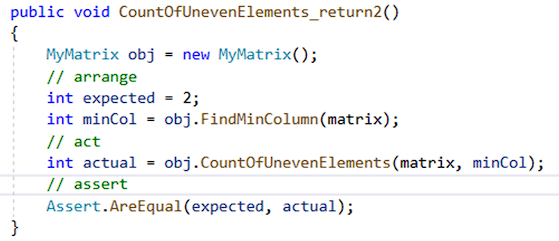
Модульное тестирование заключается в тестировании отдельного модуля, как части программы, подразумевая что это только модуль и он не может существовать самостоятельно и является частью приложения, программы.

46) Описать модульное тестирование при тестовом прогоне. Перечислить задачи модульного тестирования.

Задачи

1. Проверка правильности функционирования отдельных модулей программы.
2. Выявление дефектов и ошибок внутри модулей.
3. Проверка соответствия модулей заданным спецификациям и требованиям.
4. Проверка взаимодействия модулей друг с другом.
5. Проверка надежности и стабильности модуля.
6. Проверка корректности входных и выходных данных модуля.

47) Описать принцип создания модульного теста. Привести пример шаблона теста.



48) Описать процесс создания модульного теста для проектов, созданных С#. Перечислить фреймворки, используемые для проведения модульного тестирования.

Что-то в пр4 по процессу

**MS Test** (**Unit Test Framework)**:**–** фреймворк юнит-тестирования от компании Microsoft, который по умолчанию включен в Visual Studio и который также можно использовать с .NET Core;

**NUnit**: портированный фреймворк с JUnit для платформы .NET

49) Дать определение понятию регрессионное тестирование. Описать цели регрессионного тестирования.

Регрессионное тестирование — это вид тестирования направленный на проверку изменений, сделанных в приложении или окружающей среде (починка дефекта, слияние кода, миграция на другую операционную систему, базу данных, веб сервер или сервер приложения), для подтверждения того факта, что существующая ранее функциональность работает как и прежде.

Цели

1. Удостовериться, что программа функционирует в соответствии со своей спецификацией, что изменения не привели к внесению новых ошибок в ранее протестированный код.
2. В соответствии с используемым критерием покрытия кода (например, критерием покрытия потока операторов или потока данных), гарантировать тот же уровень покрытия что и при полном повторном тестировании программы.

50) Описать направления регрессионного тестирования. Перечислить виды регрессионных тестов.

Работа регрессионного тестирования ПО проводится по трем основным направлениям:

1. Багам
2. Старым проблемам
3. Побочным эффектам

Виды тестов

1. **Верификационные тесты.** Проводятся для проверки исправления обнаруженного и открытого ранее бага.

2. **Тестирование верификации версии.** Содержит принципы smoke test и тестирование сборки: проверка работоспособности основной функциональности программы в каждой новой сборке.

3. **Непосредственно само регрессионное тестирование.** Повторное выполнение всех тестов, которые были написаны и проведены ранее. Они выполняются по уже существующим тест-кейсам независимо от того, были в ходе их прохождения найдены баги, или нет.

4. **Тестирование в новом билде** уже исправленных багов в старых билдах. Это выполняется для того, чтобы проверить, не возобновило ли обновление билда старых дефектов.

51) Описать цели тестирования производительности. Перечислить

основные направления тестирования производительности.

1. **Оценка времени** выполнения выбранных операций при определённой интенсивности и очерёдности выполнения этих операций.
2. **Оценка реакции** приложения на изменение количества пользователей, одновременно работающих с приложением.
3. **Оценка границ интенсивности и видов нагрузки**, при которых производительность приложения выходит за рамки приемлемой.
4. **Исследование производительности** на низких, средних, высоких, предельных и стрессовых значениях нагрузки.
5. **Оценка показателей масштабируемости приложения**.

Направления

1. **Нагрузочное тестирование** (load testing)
2. **Стрессовое тестирование** (stress testing)
3. **Объемное тестирование** (volume testing)
   1. Тестирование стабильности (Stability Reliability Testing)
   2. Конфигурационное тестирование
   3. Тест на масштабируемость (scalability test)

52) Описать цели нагрузочного и стрессового тестирования производительности. Перечислить на каких стадиях проводится тестирование производительности.

Цели нагрузочного:

1. **Оценка скорости реакции** приложения на различные значения нагрузки в допустимых пределах;
2. **Оценка использования приложением системных ресурсов** при различных значения нагрузки в допустимых пределах;
3. **Оценка изменения со временем поведения** приложения при сохранении некоторой допустимой нагрузки длительное время.

Цели стрессового:

1. Оценка реакции приложения на нестандартные и стрессовые случаи изменения нагрузки в т.ч.:
   1. резкое непредсказуемое изменение интенсивности нагрузки;
   2. значительное превышение предельно допустимой нагрузки;
   3. интенсивное использование функций приложения, являющихся «узким местом» впроизводительности.

Тестирование проводится на стадии проектирования

1. **На стадии проектирования** 
   1. Какая из архитектур системы лучше, трех или четырехуровневая?
   2. Где и как хранить файлы, обеспечат ли выбранный язык и БД необходимую производительность и т.д.

53) Описать специфические особенности при тестировании мобильных

приложений.

54) Перечислить основные проблемы тестирования мобильных

приложений.

1. Мобильные устройства **работают от аккумуляторов**, и потому вынуждены автоматически переходить в режим ожидания спустя пару минут бездействия. Это значит, что придётся включать телефон перед каждым тестированием, что при одновременном тестировании нескольких телефонов, занимает приличное количество времени. Конечно, на многих устройствах можно отключить автоматическую блокировку (или хотя бы сделать время отключения довольно большим), но желательно всё же работать с самыми распространёнными среди пользователей настройками ОС.
2. Переход в **режим ожидания** особенно неприятен при использовании некоторых автоматизированных систем тестирования, требующих время для обработки данных (например, снятие и сравнение скриншотов). Возможно, что телефон перейдёт в режим ожидания прямо посреди теста.
3. Затем появляется проблема собственно набора определённого текста (например, адреса тестируемой страницы). Придётся аккуратно вводить длинные тексты в несколько устройств с разными интерфейсами.
4. Для **тестирования GPS придётся вооружиться дополнительным инструментарием** от энтузиастов и надеяться, что он работает достаточно похоже на реальные условия.
5. Для **проверки слабого или отсутствующего Wi-Fi и 3G-сигнала обычно приходится либо сооружать лабораторию**, либо использовать различные хитрости вроде коробочек из фольги.

55) Дать определение понятию автоматизированное тестирование. Описать области применения автоматизированного тестирования.

Автоматизированное тестирование – это набор техник, подходов, методов и инструментальных средств, позволяющих исключить участие человека в реализации некоторых задач по тестированию ПО.

**Области применения**

1. Проверка ссылок
2. Проверка работоспособности стандартного, типичного для множества проектов функционала

56) Описать достоинства и недостатки автоматизированного

тестирования.

Достоинства

1. **Скорость**. Да, компьютер, в общем случае, работает быстрее человека.
2. **Надёжность**. Да, компьютер допускает меньше ошибок в силу отсутствия т.н. «человеческого фактора».
3. **Мощность**. Например, человек никогда не сможет вручную добавить в БД 100миллиардов записей, состоящих из 500полей каждая.

Недостатки

1. Автоматизированное регрессионное тестирование **реже обнаруживает новые ошибки**, т.к. тесты (особенно, в случае жёстко заданных тестовых данных и путей выполнения)проходят строго по одному и тому же пути. В случае качественной работы команды разработчиков основное внимание на проекте следует уделять новой функциональности, в то время как автоматизированные тесты чаще всего направлены на проверку ранее реализованной.
2. Недостаточная продуманность вопросов поддержки автоматизированных тестов приводи к тому, что в случае серьёзных изменений в приложении **многие автоматизированные тесты становятся бесполезными** и/или требуют серьёзной переработки.
3. На автоматизацию часто возлагают неоправданные надежды, что ведёт к срывам сроков и прочим проблемам.
4. **Сложно и нецелесообразно автоматизировать проверку** графического интерфейса (верстки, расположения элементовна экране и т.д.).

57) Что представляет собой Selenium IDE. Описать процесс создания

нового теста.

Расширение к браузеру, которое позволяет

записывать, редактировать, отлаживать и выполнять тесты

59) Что представляет собой Selenium WebDriver? Классифицировать

методы WebDriver для поиска элементов.

Selenium WebDriver — универсальный интерфейс, который позволяет

манипулировать разными браузерами напрямую из кода на языке

программирования.

1. **Методы поиска по уникальному идентификатору элемента:**

* find\_element\_by\_id: Поиск элемента по уникальному идентификатору (ID).
* find\_element\_by\_name: Поиск элемента по имени.
* find\_element\_by\_class\_name: Поиск элемента по имени класса.
* find\_element\_by\_tag\_name: Поиск элемента по имени тега.

1. **Методы поиска по селекторам CSS:**

* find\_element\_by\_css\_selector: Поиск элемента с использованием CSS-селектора.

1. **Методы поиска по пути элемента:**

* find\_element\_by\_xpath: Поиск элемента с использованием языка запросов XPath.
* find\_element\_by\_link\_text: Поиск гиперссылки по полному тексту.
* find\_element\_by\_partial\_link\_text: Поиск гиперссылки по частичному тексту.

1. **Методы поиска нескольких элементов (возвращают список элементов):**

* find\_elements\_by\_id
* find\_elements\_by\_name
* find\_elements\_by\_class\_name
* find\_elements\_by\_tag\_name
* find\_elements\_by\_css\_selector
* find\_elements\_by\_xpath
* find\_elements\_by\_link\_text
* find\_elements\_by\_partial\_link\_text

60) Что представляет собой unittest? Описать алгоритм создания теста на

основе фреймворка unittest.

Unittest (или Unit Testing) - это методология тестирования программного обеспечения, при которой отдельные компоненты (юниты) программы тестируются независимо от других компонентов. Unittest является частью практики разработки программного обеспечения, известной как тестирование модульного уровня.

Алгоритм

Шаг 1: Импортирование необходимых модулей

Шаг 2: Создание класса теста

Шаг 3: Определение методов тестирования

Шаг 4: Написание утверждений (assertions)

Шаг 5: Запуск тестов

Шаг 6: Анализ результатов

Шаг 7: Доработка и повторное выполнение тестов