**Ответы на вопросы по тестированию**

**1. В каком случае в программе содержится ошибка**

Программа содержит ошибку, если она ведет себя неразумно с точки зрения пользователя.

Как определить разумность поведения:

1. Она должна быть верна синтаксически
2. Программа должна правильно решать поставленную перед ней задачу
3. Программа не должна делать ничего лишнего
4. Результат должен быть получен через разумное время при разумных затратах других ресурсов
5. Программа должна разумно реагировать на ввод некорректных данных и на предусмотренные заранее ситуации

**2. Что такое тестирование**

Тестирование – выполнение программы с целью обнаружения факта наличия в программе ошибки

**3. Что такое отладка**

Отладка – определение места ошибки и внесение исправления в программу

**4. Цель тестирования**

Цель тестирования – обнаружение ошибок в программе (не доказательств ее правильности!)

**5. Принципы тестирования**

1. Ошибки в программе есть
2. Тест – это совокупность исходных данных и ожидаемых результатов
3. Тестовые данные должны быть достаточно просты для проверки
4. Тесты готовятся заранее до выхода на машину
5. Первые тесты разрабатываются после получения задания на разработку программы до написания программного кода
6. Перед началом тестирования следует сформулировать цели, которые должны быть достигнуты в ходе тестирования
7. В процессе тестирования необходимо фиксировать выполненные тесты и реально полученные результаты
8. Тесты должны быть одинаково тщательными как для правильных, так и для неправильных входных данных
9. Необходимо проверить два момента: программа делает то, что должна делать; программа не делает того, чего делать не должна
10. Результаты теста необходимо изучать досконально и объяснять полностью
11. Недопустимо ради упрощения тестирования изменять программу
12. После исправления программы необходимо повторное тестирование
13. Ошибки “кучкуются”
14. Окончательное тестирование программы лучше проводить не ее автору, а другому человеку.

**6.** **Таблица тестирования**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № | Входные данные | Ожидаемые результаты | Результаты сухой проверки | +/- | Реальные результаты | +/- | Коментарии |

**7. Полнота тестирования**

Исчерпывающее тестирование для любой нетривиальной программы невозможно. Для уменьшения количества проверяемых вариантов все множество возможных вариантов выполнения программы делят на подмножества (классы). Все варианты, попавшие в одно подмножество, считаются в некотором смысле равнозначными. Считается, что, проверив один из вариантов данного подмножества, мы тем самым проверим и все остальные варианты данного подмножества. Критерии, по которым проводится классификация всех возможных вариантов выполнения программы с точки зрения проверки ее правильности, называются **критериями полноты тестирования.** Для повышения надежности проводят несколько классификаций, делят множество вариантов выполнения на классы несколькими разными способами.

**8. Отличие критериев Черного ящика и Белого ящика**

Критерии **черного ящика** описывают тестирование с точки зрения поставленной задачи без учета внутреннего устройства программы. Критерии **белого ящика** учитывают структуру программы.

**9. Критерии Черного ящика**

1. Тестирование функций
2. Тестирование классов входных данных.
3. Тестирование классов выходных данных
4. Тестирование области допустимых значений (границ класса)
5. Тестирование длинны набора данных
6. Тестирование упорядоченности набора данных

**10. Тестирование функций**

Данный критерий актуален для многофункциональных программ. Он требует подобрать такой набор тестов, чтобы был выполнен хотя бы один тест для каждой из функций, реализуемых программой.

**11. Тестирование классов входных данных**

Данный критерий требует классифицировать входные данные, разделить их на классы таким образом, чтобы все данные из одного класса были равнозначны с точки зрения проверки программы. Считается, что если программа работает правильно на одном наборе входных данных из этого класса, то она будет правильно работать на любом другом наборе данных из этого же класса. Критерий требует выполнения хотя бы одного теста для каждого класса входных данных.

**12. Тестирование классов выходных данных**

Этот критерий выглядит аналогично предыдущему, только проверяются не входные, а выходные данные.

**13. Тестирование ОДЗ (границ класса)**

Если ОДЗ переменной представляет собой **простое перечисление**, надо проверить сто программа правильно понимает все эти значения и не принимает вместо них никаких других значений. Если класс допустимых значений представляет собой **числовой диапазон**, то понадобится более серьезная проверка. В этом случае выделяются:

1. Нормальные условия (в середине класса)
2. Граничные (экстремальные) условия
3. Исключительные условия (выход за границу класса).

**14. Тестирование длинны набора данных**

Можно считать частным случаем тестирования ОДЗ. В данном случае речь идет о допустимом количестве элементов в наборе. Если программа последовательно обрабатывает элементы некоторого набора данных, имеет смысл проверить следующие ситуации:

1. Пустой набор (не содержит ни одного элемента)
2. Единичный набор (состоит из одного единственного элемента)
3. Слишком короткий набор (если предусмотрена минимально допустимая длинна)
4. Набор минимально возможной длинны (если такая предусмотрена)
5. Нормальный набор (состоит из нескольких элементов)
6. Набор из нескольких частей (если такое возможно)
7. Набор максимально возможной длинны (если она предусмотрена)
8. Слишком длинный набор (с длинной больше максимально допустимой)

**15. Тестирование упорядоченности набора данных**

Важно для задач сортировки и поиска экстремумов. Имеет смысл проверить следующие ситуации:

1. Данные не упорядочены
2. Данные упорядочены в прямом порядке
3. Данные упорядочены в обратном порядке
4. В наборе имеются повторяющиеся значения
5. Экстремальные значения находятся в середине набора
6. Экстремальные значения находятся в начале набора
7. Экстремальные значения находятся в конце набора
8. В наборе несколько совпадающих экстремальных значений

**16. Тестирование поиска в наборе данных**

**17. Таблица для проверки полноты тестирования методами Черного ящика**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Критерий | Тесты | | | | | |  | | № Теста | | Входные данные | | Ожидаемый результат | Проверяет | |
| Т1 | Т2 | Т3 | Т4 | Т5 | Т6 | |  | |  | |  |  |
| 1\_\_\_\_\_\_\_ |  |  |  |  |  |  | |  | | Т1 | |  |  | |  |
| 1.1\_\_\_\_\_\_ |  |  |  |  |  |  | |  | | Т2 | |  |  | |  |
| 1.2\_\_\_\_\_ |  |  |  |  |  |  | |  | | Т3 | |  |  | |  |

**18. Перечислить критерии белого ящика**

1. Покрытие операторов
2. Покрытие ветвей (решений)
3. Покрытие путей
4. Покрытие условий
5. Покрытие решений / условий
6. Комбинаторное покрытие условий

**19. Покрытие операторов**

Требует подобрать такой набор тестов, чтобы каждый оператор в программе был выполнен хотя бы один раз.

**20. Покрытие решений**

Требует подобрать такой набор тестов, чтобы каждая ветвь в программе была выполнена хотя бы один раз.

**21. Покрытие путей**

Требует подобрать такой набор тестов, чтобы каждый путь в программе был выполнен хотя бы один раз.

**22. Покрытие условий. Комбинаторное покрытие условий**

Критерий покрытия условий требует подобрать такой набор тестов, чтобы каждое простое условие получило значение и “истина” и “ложь” хотя бы один раз.

Критерий комбинаторного покрытия условий требует подобрать такой набор тестов, чтобы хотя бы один раз выполнялась любая комбинация простых условий.

**23. Покрытие решений / условий**

Требует подобрать такой набор тестов, чтобы каждая ветвь в программе была пройдена хотя бы один раз и чтобы каждое простое условие получило значение и “истина” и “ложь” хотя бы один раз.

**24. МГТ для if с простым условием и для if со сложным условием**

МГТ – минимально грубое тестирование. Представляет собой критерий покрытия решений / условий, усиленный дополнительными требованиями по проверке циклов.

Для критерия МГТ существует очень удобное представление в форме таблицы. Строки таблицы соответствуют проверяемым условиям, графы – тестам.

Для каждого условного оператора в таблице МГТ создаются 2 строки: для ветви “то” и ветви “иначе”

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | Тест 1 | Тест 2 | Тест 3 | Тест 4 | Тест 5 |
| If a > b | + |  |  |  |  |  |
| - |  |  |  |  |  |

Если в условном операторе, в цикле с пред – или постусловием стоит сложное условие, то к строкам, соответствующим самому оператору, добавляются по 2 строки на каждое простое условие.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | Тест 1 | Тест 2 | Тест 3 | Тест 4 | Тест 5 |
| If (a > b) and (x = y) | + |  |  |  |  |  |
| - |  |  |  |  |  |
| a > b | + |  |  |  |  |  |
| - |  |  |  |  |  |
| x > y | + |  |  |  |  |  |
| - |  |  |  |  |  |

**25. МГТ для case**

Для каждого оператора выбора записывается столько строк, сколько он имеет ветвей (включая ветвь “иначе”).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | Тест 1 | Тест 2 | Тест 3 | Тест 4 | Тест 5 |
| case color | red |  |  |  |  |  |
| green |  |  |  |  |  |
| blue |  |  |  |  |  |
| Else |  |  |  |  |  |

**26. МГТ для цикла while с простым и сложным условием**

Для каждого цикла с предусловием – 3 строки: для нулькратного, однократного и многократного повторения тела цикла.

Таблица МГТ для цикла while с простым условием:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | Тест 1 | Тест 2 | Тест 3 | Тест 4 | Тест 5 |
| while a > b | = 0 |  |  |  |  |  |
| = 1 |  |  |  |  |  |
| > 1 |  |  |  |  |  |

Таблица МГТ для цикла while со сложным условием:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | Тест 1 | Тест 2 | Тест 3 | Тест 4 | Тест 5 |
| while (c > d) or (z = f) | = 0 |  |  |  |  |  |
| = 1 |  |  |  |  |  |
| > 1 |  |  |  |  |  |
| c > d | + |  |  |  |  |  |
| - |  |  |  |  |  |
| z = f | + |  |  |  |  |  |
| - |  |  |  |  |  |

**27. МГТ для цикла repeat**

Для каждого цикла с постусловие – 2 строки: для однократного и многократного повторения тела цикла

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | Тест 1 | Тест 2 | Тест 3 | Тест 4 | Тест 5 |
| repeat until a > b | = 1 |  |  |  |  |  |
| > 1 |  |  |  |  |  |

**28. МГТ для цикла for**

Для каждого цикла со счетчиком – 3 строки: для нулькратного, однократного и многократного повторения тела цикла.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | Тест 1 | Тест 2 | Тест 3 | Тест 4 | Тест 5 |
| For k: = 1  to n | = 0 |  |  |  |  |  |
| = 1 |  |  |  |  |  |
| > 1 |  |  |  |  |  |

**29. Ошибкоопасные ситуации при обращении к данным**

1. Использование значения переменной. Опасность: неинициализированная переменная. Переменная используется до того как ей присваивается значение
2. Автоматическая инициализация переменных. Опасность: неверная инициализация
3. Индексация массива. Опасность: выход индекса за границу измерения
4. Изменение переменной внутри блока. Опасность: побочный эффект – изменение глобальной переменной при выполнении подпрограммы
5. Использование значения ссылочной переменной. Опасность: “висячая ссылка”
6. Схожие имена переменных
7. Использование записи с вариантами. Опасность: несколько полей записи с вариантами используются для обращения к одной и той же области памяти при отсутствии контроля за правильной типизацией.
8. Использование не типизированного указателя. Опасность: еще одна возможность обойти типовой контроль. Динамическая переменная, на которую ссылается данный указатель в разных местах программы, может трактоваться как переменная разных типов
9. Обращение к одной и той же области памяти из разных модулей.
10. Использование статических переменных, использование типизированных констант. Опасность: путаница между статическими и динамическими переменными
11. Использование переменных не имеющих явного описания.

**30. Ошибкоопасные ситуации при вычислениях**

1. Выражение. Опасность: неверный порядок вычисления операций в выражении
2. Логическое выражение. Опасность: путаница между полной и краткой схемами вычисления выражений
3. Сравнение > / <. Опасность: Потеря третьего результата операции сравнения
4. Сравнение > и ≥ , < и ≤. Опасность: ошибки типа “± 1”
5. Деление. Опасность: деление на нуль
6. Извлечение квадратного корня. Опасность: корень из отрицательного числа
7. Взятие логарифма. Опасность: логарифм неположительного числа.
8. Использование в вычислениях данных “не того” типа. Опасность: неверное приведение типов данных
9. Присваивание целой переменной вещественного значения. Опасность: способ преобразования вещественного числа в целое
10. Вычисления с плавающей точкой (вещественная арифметика). Опасность: погрешности округления, потеря значимости (получение чисел с очень маленькой мантиссой)
11. Сравнение вещественных чисел. Опасность: погрешности округления
12. Арифметические вычисления (как вещественные, так и целые). Опасность: переполнение (получение очень больших чисел), переполнение или потеря значимости в промежуточных вычислениях

**31. Ошибкоопасные ситуации при передаче управления**

1. Развилки. Опасность: пропущена ветвь “иначе”
2. Вложенные условные операторы. Опасность: сочетание if – then – if – then – else
3. Циклы. Опасность: зацикливание

**32. Ошибкоопасные ситуации при вызовах программ**

1. Вызов подпрограммы. Опасность: неверное количество параметров, неверные типы параметров, неверный порядок следования параметров
2. Использование параметров внутри подпрограммы. Опасность: неверные единицы измерения параметров
3. Использование формального параметра внутри подпрограммы для измерения значения соответствующего фактического параметра. Опасность: неверный способ передачи параметров
4. Использование внутри подпрограммы формальных параметров, передаваемых по ссылке. Опасность: одна и та же переменная может быть указана как фактический параметр одновременно для нескольких формальных параметров передаваемых по ссылке
5. Использование внутри подпрограммы глобальной переменной и параметра, передаваемого по ссылке. Опасность: одновременный доступ к одной и той же переменной через параметр, передаваемый по ссылке, и через глобал.

**33. Ошибкоопасные ситуации при работе с файлами**

1. Запись / чтение двоичных файлов. Опасность: путаница между файлами разных типов
2. Создание файлов с данными с помощью текстового редактора. Просмотр файлов с данными с помощью текстового редактора. Опасность: путаница текстовых и двоичных файлов
3. Запись / чтение не типизированных файлов. Опасность: данные могут быть записаны в файл как значение одного типа, а прочитаны как значения другого
4. Запись в файл. Опасность: отсутствие явного закрытия файлов.

**34. Виды безмашинного тестирования**

1. Сухая прокрутка
2. Символическая прокрутка
3. Объяснения коллеге
4. Передача программы коллеге для самостоятельного изучения
5. Просмотр текста программы с анализом ошибкоопасных мест
6. Искусственное внесение ошибок в программу.

**35. Что такое заглушка**

Заглушка представляет собой имитатор еще не разработанной части программы. Она может быть подставлена вместо подпрограммы или фрагмента, написанного на псевдокоде.

**36. Виды заглушек**

В качестве заглушек могут использоваться: пустые операторы, операторы печати, упрощенные варианты имитируемых частей.

**38. Достоинства нисходящего тестирования**

1. Возможность раннего начала тестирования
2. Ранняя проверка сопряжений между модулями
3. Более тщательная проверка модулей верхнего уровня

**39. Недостатки нисходящего тестирования**

1. Трудно тестировать модули нижнего уровня
2. Необходимо тратить силы на написание заглушек
3. Тестирование на заглушках оказывается неполным, после замены их реальным программным текстом тестирование приходится повторять
4. Заглушки можно подставить только для процедур и операторов, но не для переменных и типов.

**40 Оценка количества ошибок в программе: модель Миллса.**

Условия:

* Имеется программа, в которой N естественных ошибок
* Внесем дополнительно M искусственных ошибок
* В ходе тестирования обнаружено n – естественных и m – искусственных ошибок, предположим, что вероятность обнаружения и тех и других одинакова т.е. выполняется соотношение

Мы нашли один и тот же процент искусственных ошибок → количество ошибок в программе Количество обнаруженных ошибок равно

Существенный недостаток модели: если находим 100 % искусственных ошибок, это будет означать, что найдены 100 % естественных ошибок. Чтобы исключить данный недостаток Миллс ввел вторую часть модели, которая проверяет гипотезу о величине *N* .

**41. Мера доверия к миллсовой модели**

Условия:

* Имеется программа, в которой N естественных ошибок
* Внесем дополнительно M искусственных ошибок и будем тестировать программу пока не найдем все искусственные ошибки. Пусть к этому моменту найдено *n* естественных ошибок.

Тогда

Величина *C* выражает меру доверия к модели. Это вероятность того, что модель будет правильно отклонять ложное предположение. Верхняя строка 100 % опровергает гипотезу о том, что естественных ошибок в программе не более чем *N* (предположили *N,* обнаружили больше). Для получения нужной уверенности в полученной оценке можно рассчитать количество необходимых искусственных ошибок:

Слабое место модели Миллса – предположение о равновероятности нахождения ошибок.

**42. Оценка количества ошибок в программе: «парная оценка».**

Условия:

* Программу тестируют независимо друг от друга две группы специалистов
* В программе содержится N ошибок. Первая группа обнаружила ошибок, вторая - Часть ошибок обнаружена обеими группами
* Обнаружение всех ошибок считаем равновероятным

Тогда: Доля всех ошибок, найденная первой группой = . Доля ошибок, найденная первой группой среди ошибок, найденных второй группой = .

Отсюда количество ошибок в программе:

Количество найденных ошибок =

**43. Место проявления ошибки и место нахождения ошибки**

Необходимо различать место проявления и место нахождения ошибки. По каким-то внешним признакам мы можем найти только первое. Второе можно определить лишь путем содержательного анализа текста программы.

**44, 45 Отладочные операторы. Условная отладка**

Как правило, отладочные операторы – это операторы печати, которые выполняют полную или частичную трассировку (пошаговая отладка) значений нужных переменных. Трасса может выдаваться на экран или записываться в файл.

Еще один полезный инструмент отладки – встраивание в текст программы утверждений о том, каким ограничениям должны удовлетворять те или иные переменные.

Резко усилить мощь отладочных средств можно сделав их условными. В программе описываются одна или несколько специальных переменных для управления отладкой (отладочные флаги). Отладочные операторы помещаются внутрь условных операторов, выполнение которых зависит от отладочных флагов.Это позволяет включать и выключать отладочные действия без изменения текста программы.

Платой за наличие отладочных операторов в тексте программы станет некоторое повышение требований к машинным ресурсам. Отладочные операторы увеличивают размер исходной программы, время трансляции, размер кода, время выполнения. Избежать этого можно сделав операторы условно транслируемыми. После удаления из программы отладочных операторов ее необходимо вновь протестировать.

**46. Индуктивный поиск ошибки**

Индуктивный подход означает движение от частного к общему. От того, какие данные говорят об ошибке, к тому, как их можно объяснить.

Схема индуктивных рассуждений

Сбор данных о работе программы

Выделение данных, имеющих отношение к ошибке

Изучение зависимостей между этим данными

Выдвижение гипотезы об ошибке

Проверка гипотезы

Гипотеза

Подтвердилась?

Нет

Да

Сбор данных о работе программы

**47. Дедуктивный поиск ошибки**

Дедуктивный подход означает движение от общего к частному. Исходя из некоторых общих соображений формируется множество гипотез. Затем оно уточняется: какие-то гипотезы исключаются, какие-то уточняются за счет дополнительной информации.

Схема дедуктивных рассуждений

Перечисление гипотез

Исключение и уточнение гипотез

Исключаются все гипотезы?

Да

Сбор дополнительных данных

Нет

Проверка выбранной гипотезы

Гипотеза

подтвердилась?

Нет

Да

Нахождение ошибки

**48. Ретроанализ**

Находим место ошибки и прослеживаем по программе назад откуда она взялась.

**49. Принципы отладки**

1. Думать. Средства отладки играют только вспомогательную роль
2. Избегать экспериментирования
3. Исправлять поочередно
4. Необходимо найти ошибку, которая объясняла бы 100 % симптомов
5. Там, где есть ошибка, может быть еще
6. Исправление может внести новую ошибку.

**50. Анализ обнаруженной ошибки**

Ошибки следует анализировать по следующему плану:

1. Когда была сделана ошибка?
2. Почему была сделана ошибка?
3. Как можно было предотвратить ошибку?
4. Почему ошибку не обнаружили раньше?
5. Как можно было обнаружить ошибку раньше?

Результаты анализа полезно фиксировать в “дневнике отладки”

**51. Роли участников тестирования**

1. Тест – аналитик
2. Тест – дизайнер
3. Тестировщик

**52. Роль “тест – аналитик”**

Отвечает на вопрос: что надо тестировать

**53. Роль “тест – дизайнер”**

Ответственен за разработку конкретных тестов. Отвечает на вопрос: Как надо тестировать

**54. Роль “тестировщик”**

Ответственен за непосредственное тестирование компонента или системы. В обязанности тестировщика входит поиск вероятностных ошибок и сбоев в функционировании объекта тестирования.

**55. Описание “тест – кейса”**

Тест-кейс – это профессиональная документация тестировщика, последовательность действий направленная на проверку какого-либо функционала, описывающая как прийти к фактическому результату. Набор тест-кейсов называется тестовым набором (test suite).

Стандартные атрибуты тест-кейса:

1. Номер – уникальный идентификатор тест-кейса
2. Название – краткое описание сути проверки
3. Предварительные шаги (необязательно)
4. Шаги – описание действий, необходимых для проверки
5. Ожидаемый результат – сама проверка, что мы ожидаем получить после выполнения шагов.

**56. Позитивный тест**

Позитивное тестирование – это тестирование с применением сценариев, которые соответствуют нормальному (штатному, ожидаемому) поведению системы. С его помощью мы можем определить, что система делает то, для чего и была создана.

**57. Негативный тест**

Негативное тестирование – тестирование в рамках которого применяются сценарии, несоответствующие штатному поведению тестируемой системы (исключительные ситуации, неверные данные). Негативное тестирование направлено на проверку устойчивости системы к различным воздействиям, валидации неверных данных, обработке исключительных ситуаций.

**58. Подходы к тестированию**

1. Скриптовое тестирование
2. Исследовательское тестирование
3. Сессионное тестирование

**59. Суть скриптового тестирования**

Заблаговременной написание тестирующей программы (скрипта)

**60. Суть исследовательского тестирования**

Ничего заранее не продуманно, полная импровизация.

**61. Суть сессионного тестирования**

Тестирование модулей программы

**62. Достоинства скриптового тестирования**

* Четкое разделение ролей (тест-аналитик, тест-дизайнер, тестировщик)
* Отделение высококвалифицированного труда от неквалифицированного
* Возможность многократного повторения теста
* Возможность получить толковый отчет
* Можно проанализировать покрытие кода
* Обеспечение единообразия понимания того, что должна делать программа.

**63. Недостатки скриптового тестирования**

* Пока скрипта нет тестирование не начнешь
* Написание скрипта занимает время
* Затраты на поддержание скрипта в актуальном состоянии возрастают с внесением изменений в тестируемую программу.

**64. Достоинства исследовательского тестирования**

* Меньше временных затрат благодаря тому, что не требуется специальной подготовки, а успех в большей степени зависит от когнитивного мышления тестировщика
* Опора на имеющийся опыт тестирования при поиске ошибок

**65. Недостатки исследовательского тестирования**

* Один человек выполняет все роли → использование квалифицированного специалиста в роли тестера
* Некоторые важные дефекты могут быть пропущены

**66. Достоинства сессионного тестирования**

* Компромисс между двумя крайностями – управляемостью скриптового тестирования и творчеством исследовательского тестирования, усредненный вариант
* Есть точки тестирования, с которыми ведется работа на протяжении одной сессии, имеется небольшой объем документации
* Гораздо больше свободы, чем в случае с подробным скриптовым тестированием.

**67. Недостатки сессионного тестирования**

* Нет детальной регламентации, как в скриптовом тестировании
* Сессионное тестирование предполагает совместный разбор полетов, на что не всегда хватает времени
* Во время сессии не следует прерываться, но бывают форс-мажорные ситуации.

**68. Эффект пестицида**

Повторное применение одних и тех же тестов, методов тестирования, приводит к тому, что в программе остаются дефекты, против которых эти методы не эффективны. Термин ввел Бориз Бейзер в своей книге “Software Testing Techniques” в 1983 г.

**69. Виды тестирования**

Функциональные виды тестирования, нефункциональные виды тестирования, виды тестирования связанные с изменением программы.

**70. Виды тестирования связанные с изменением программы**

1. Тестирование сборки
2. Дымовое тестирование
3. Санитарное тестирование
4. Регрессионное тестирование.

**71. Тестирование сборки**

Тестирование, направленное на определение соответствия выпущенной версии критериям качества для начала тестирования.

**72. Дымовое тестирование**

Короткий цикл тестов, выполняемый для подтверждения того, что после сборки кода (нового или исправленного) программа в принципе работает.

**73. Санитарное тестирование**

Узконаправленное тестирование, достаточное для доказательства того, что конкретная функция работает согласно заявленным в спецификации требованиям.

**74. Регрессионное тестирование**

Вид тестирования, направленный на проверку изменений, сделанных в приложении или окружающей среде, для подтверждения того факта, что существующая ранее функциональность работает, как и прежде. Есть три типа регрессионного тестирования:

-регрессия дефектов,

-регрессия старых дефектов,

- регрессия побочного эффекта.

**75. Регрессия дефектов**

Попытка доказать что исправленная ошибка на самом деле не исправлена

**76. Регрессия старых дефектов**

Попытка доказать что недавнее изменение кода или данных сломало исправление старых ошибок, т.е. старые баги стали снова воспроизводиться.

**77. Регрессия побочного эффекта**

Попытка доказать что недавнее изменение кода или данных сломало другие части разработанного приложения.

**78. Виды нефункционального тестирования**

1. Тестирование юзабилити
2. Коридорное Тестирование
3. Пятисекундное тестирование
4. Тестирование безопасности
5. Тестирование локализаций
6. Тестирование производительности
7. Стрессовое тестирование
8. Объемное тестирование
9. Тестирование стабильности
10. Конфигурационное тестирование
11. Инсталяционное тестирование
12. Тестирование документации

**79. Тестирование юзабилити**

Метод тестирования направленный на установление степени удобства использования, обучаемости, понятности и привлекательности для пользователей разрабатываемого продукта в контексте заданных условий. Тестирование удобства использования дает оценку уровня удобства использования приложения по следующим пунктам: производительность, эффективность, правильность, активизация в памяти, эмоциональная реакция.

**80. Коридорное тестирование**

Создается макет программы и демонстрируется коллегам. Задаются вопросы:

1. Куда бы ты нажал, чтобы сделать / найти что-то.
2. Как ты думаешь, что произойдет, если ты нажмешь на эту кнопку.
3. Как ты понимаешь эту фразу / название.

**81. Пятисекундное тестирование**

Макет программы демонстрируется человеку в течении 5 секунд. Затем макет убирается и человеку задается вопрос о том, что он увидел / запомнил на макете.

**82. Тестирование безопасности**

Стратегия тестирования, используемая для проверки безопасности системы, а также для анализа рисков, связанных с обеспечением целостного подхода к защите приложения от хакеров, вирусов, несанкционированного доступа к конфиденциальным данным. Общая стратегия безопасности основывается на трех принципах: конфиденциальность, целостность, доступность.

**83. Тестирование локализации**

Процесс тестирования локализованной версии программного продукта или сайта. Локализация – это процесс адаптации к культурным особенностям той или иной страны: перевод документации, элементов пользовательского интерфейса, вспомогательных материалов с одного языка на другой.

**84. Тестирование производительности**

Автоматизированное тестирование, имитирующее работу определенного количества бизнес-пользователей на каком-либо общем (разделяемом ими) ресурсе. Включает в себя тестирование производительности, стрессовое тестирование, объемное тестирование и тестирование стабильности / надежности. Задачей тестирования производительности является определение масштабируемости приложения под нагрузкой. При этом происходит:

* Измерение времени выполнения операций при определенных интенсивностях их выполнения
* Определение количества пользователей одновременно работающих с приложением
* Определение границ приемлемой производительности при увеличении нагрузки
* Исследование производительности на высоких, предельных, стрессовых нагрузках

**85. Стрессовое тестирование**

Стрессовое тестирование позволяет проверить, насколько приложение и система в целом работоспособны в условиях стресса и оценить способность системы к регенерации, т.е. возвращению к нормальному состоянию после прекращения воздействия стресса.

**86. Объемное тестирование**

Задачей является получение оценки производительности при увеличении объемов данных в БД приложения, при этом происходит:

* измерение времени выполнения выбранных операций при определенных интенсивностях выполнения этих операций
* может производиться определение количества пользователей, одновременно работающих с приложением.

**87. Тестирование стабильности**

Задачей является проверка работоспособности приложения при длительном тестировании со средним уровнем нагрузки.

**88. Конфигурационное тестирование**

Вид тестирования направленный на проверку работы ПО при различных конфигурациях системы. Может иметь разные цели:

* Определить оптимальную конфигурацию оборудования, обеспечивающую требуемые характеристики производительности и времени реакции тестируемой системы
* Проверить объект тестирования на совместимость с объявленным в спецификации оборудованием, ОС и программными продуктами третьих фирм.

**89. Инсталяционное тестирование**

Направлено на проверку успешной инсталяции и настройки, а также обновления или удаления ПО.

**90. Тестирование документации**

Проверка соответствия документации определенным требованиям. Перечень качеств, на которые следует ориентироваться при тестировании документации:

* работоспособность сценариев
* полнота описания
* уделение внимание обязательным пунктам
* актуальность описания
* адаптированность к тому, что пользователь будет в спешке / раздражен
* структурированность адаптированность к быстрому поиску
* наличие указания на необратимость действий
* подтверждение ожидаемого
* описание последствий отсутствия действий пользователя
* ясность изложения информации
* логика и согласованность
* последовательность изложения
* орфография, синтаксис, пунктуация
* наличие описания настроек по умолчанию
* адаптированность к аудитории
* атомарность сценариев
* адаптированность к наименее возможной квалификации пользователей

**91. Оценка трудозатрат на тестирование**

Оценка трудозатрат, помимо получения данных непосредственно о необходимом количестве персонала, также позволяет определить приблизительную стоимость, сроки тестирования, примерный график работ. Существуют различные методы проведения такой оценки:

* Метод “пальцем в небо”
* Экспертная оценка
* Специальный метод
* Структура декомпозиции работ
* Метод Дельфи
* Метод определения трудозатрат в процентном отношении к разработке
* Метод процентного распределения

При осуществлении оценки трудозатрат имеют также значение уровень мастерства команды, наличие и качество проектной документации, применение автоматизации, используемые инструменты и средства при тестировании.

**92. Гиперкуб для планирования проверки ПП**

**93. Уровни тестирования**

1. Компонентное (модульное)
2. Интеграционное
3. Системное
4. Приемно-сдаточные испытания

**94. Компонентное тестирование**

Проверяет функциональность и ищет дефекты в частях приложения, которые доступны и могут быть протестированы по-отдельности (модули, объекты, классы, функции и т.д.)

**95. Test-driven development**

Test first. No test, no code. Сначала пишутся тесты, которые будут применяться для проверки программы.

**96. 97. Интеграционное тестирование. Походы к интеграционному тестированию**

Предназначено для проверки связи м/д компонентами, а также взаимодействия с различными частями системы. Уровни интеграционного тестирования: компонентный и системный интеграционный уровень.

Подходы к интеграционному тестированию: с заглушками, без заглушек, снизу вверх, сверху вниз, “большой взрыв”

**98. Системное тестирование**

Проверка функциональных и нефункциональных требований к системе. При этом выявляются такие дефекты, как неверное использование ресурсов системы, непредусмотренные комбинации данных пользовательского уровня, несовместимость с окружением, непредусмотренные сценарии использования, отсутствующая или неверная функциональность, неудобство использования и т.д.

**99. Подходы к системному тестированию**

Можно выделить два подхода:

1. На базе требований – для каждого требования пишутся тестовые случаи, проверяющие их выполнение
2. На базе случаев использования – на основе представления о способах использования продукта создаются случаи использования системы, по конкретному случаю использования можно определить один или более сценариев, на проверку каждого из которых пишутся тест кейсы.

**100. Unit – тестирование**

Процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы. Тесты пишутся для каждой нетривиальной функции или метода.

**101. Классы Assert в C# Visual Studio**

Assert – это специальная конструкция позволяющая проверить предположение о значениях произвольных данных в произвольном месте программы. Она может автоматически оповещать при обнаружении некорректных данных, что обычно приводит к аварийному завершению программы с указанием места обнаружения некорректных данных. Assert’ ы можно разделить на следующие классы:

* Проверка входящих аргументов в начале функции
* Проверка данных с которыми работает функция внутри кода функции.