Вопросы на собеседовании

|  |
| --- |
| **ВОПРОС:** *В чем отличие тест-кейса от чеклиста?* |

**ОТВЕТ:** ***Чек лист*** – это список проверок, которые необходимо провести для проверки продукта, а тест-кейс - это конкретные шаги, которые нужно выполнить для этих проверок.

Чек лист каждый сотрудник может составить на свое усмотрение, а тест кейс – это шаги, которые конкретно прописаны для выполнения. Чек лист можно поменять, а тест-кейс нет.

|  |
| --- |
| **ВОПРОС:** *Когда нужно выбирать чеклист, а когда тест-кейс?* |

**ОТВЕТ:** Тест кейс стоит выбирать для работы джуна, а чек лист удобен для опытных сотрудников. Тест кейс необходим для разработки особо важных программ, например военная сфера, медицина и т.п, а чек лист в таких продуктах не подойдет.

|  |
| --- |
| **ВОПРОС:** Отличие высокоуровневого от низкоуровневого тесткейса? |

**ОТВЕТ:** ***Высокоуровневые тесткейсы*** - кейсы, которые описывают поведение похожие на чеклисты, где шаги можно додумать. При отсутствии времени.

Низкоуровневые тесткейсы - кейсы, в которых шаги подробно описаны и каждые шаги хорошо проработаны. Занимают много времени.

|  |
| --- |
| **ВОПРОС:** Отличие Test Suite от Test Case? |

**ОТВЕТ:** ***Test Suite*** - тестовый набор, который включает в себя набор Test Case.

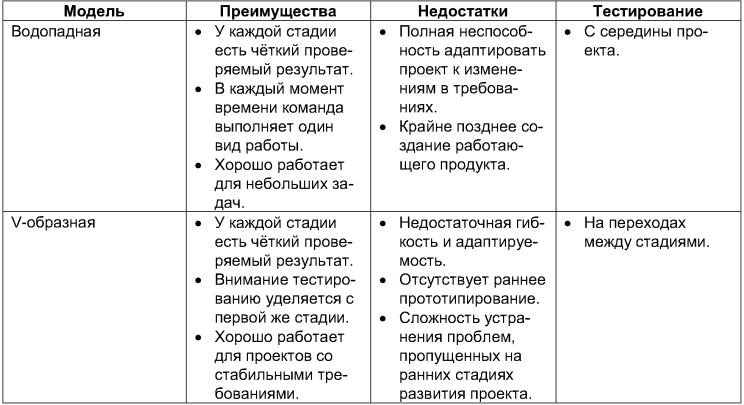
***Test Case*** - это конкретные шаги, которые нужно выполнить для проверок. (набор входных значений, предусловий, результатов, постусловий для выполнения определенного пути программы)

**Обеспечение качества (Quality Assurance)** - комплекс мероприятий направленный на обеспечение качества разрабатываемого продукта, на всех стадиях.

|  |
| --- |
| Делится на: |
| 1. Контроль качества - различные меры на стадиях разработки. Улучшение и мониторинг при разработке ПО. |
| 2. Тестирование ПО - проверка соответствия между реальным и ожидаемым поведением программы. |

**МОДЕЛИ РАЗРАБОТКИ:**

1. Водопадная модель
2. V – образная модель
3. Инкрементальная модель
4. Спиральная модель
5. Гибкая модель (SCRUM)
6. Модель Хаоса





**ОБЪЕКТЫ ТЕСТИРОВАНИЯ:**

1. Documentation
2. Prototype
3. Hardware
4. Software: сайты, десктоп, интернет вещи (Умный дом, часы, холодильник», чат боты), игры, мобильные, другое (CRM
5. Code

**ТРЕБОВАНИЯ:**

**Требования** — это спецификация (описание) того, что должно быть реализовано. Требования описывают то, что необходимо реализовать, без детализации технической стороны решения.

1. Прямые (ссылка на сайт)
2. Косвенные (что-то должно открыться, но неизвестно что)

**ТИПЫ ТРЕБОВАНИЙ:**

|  |  |
| --- | --- |
| **Функциональные** | **Не Функциональные** |
| Бизнес требования | Производительность |
| Требования пользователей | Интерфейсы работы |
| Функциональные требования |  |

**ИСТОЧНИКИ ТРЕБОВАНИЙ:**

|  |  |  |
| --- | --- | --- |
| Заказчик | Мозговой штурм | Документы |
| Фокус Группа | Наблюдение | Моделирование |
| Анкетирование | Прототип | Описание |
| Нормы | Лучшие практики | Конкуренты |

**ХАРАКТЕРИСТИКИ ТРЕБОВАНИЙ:**

|  |
| --- |
| Завершенность (завершенность главных целей) |
| Непротиворечивость (не должно быть противоречий при выполнении операций) |
| Корректность (корректное требование к программе, а не к пользователю) |
| Недвусмысленность (программа передает большие данные – какие? Уточнение) |
| Проверяемость (должна быть возможность проверить требование) |
| Атомарность (1 требование – для одного действия) |
| Выполнимость (все должно быть выполнимо) |
| Обязательность (должны быть используемые, полезные ресурсы) |

**ТЕСТ ПЛАН:**

|  |  |  |
| --- | --- | --- |
| **Мастер** | **План** | **Приемо-сдаточный** |
| Требования к дефектам | Перечень работ |  |
| Условия принятия сборки | Критерии качества |  |
| Кр-рии принят. к релизу | Оценка рисков |  |
| Инструменты | Документация |  |
|  | Стратегия |  |
|  | Ресурсы |  |
|  | Расписание |  |
|  |  |  |
|  |  |  |

**ТЕСТИРОВАНИЕ ПО СТЕПЕНИ АВТОМАТИЗАЦИИ:**

|  |  |  |
| --- | --- | --- |
| Ручное | Полуавтоматизированное | Автоматизированное |

**ВИДЫ ТЕСТИРОВАНИЯ ПО ОБЪЕКТАМ:**

|  |
| --- |
| **Функциональное** |
| Функциональное тестирование |
| GUI тестирование элемента интерфейса |
| Безопасность |

**ФУНКЦИОНАЛЬНОЕ ТЕСТИРОВАНИЕ**

|  |
| --- |
| Требования |
| Входные данные |
| Выходные данные |
| Прохождение сценариев |
| Сравнение результатов |

|  |
| --- |
| **Не функциональное** |
| **Тестирование интерфейса (UI)** |
| **Юзабилити тестирование**  - Как быстро пользователь достигнет цели  - Как долго вспоминать то, чему научился  - Размер кнопок (удобно ли попадать)  - Часто ли пользователю нужно выбирать  - Большие ли списки  - Порог вхождения |
| **Локализация**  - текст: смысловая нагрузка, синтаксис, грамматика, пунктуация, наличие текста  - Функционал: формат времени, даты, валюта, региональные настройки, часовой пояс, календарь (буддийский, японский, григорианский), начало недели, формат бумаги, система мер, особенности имен, сортировка.  - Региональные особенности: толкование текста, символов, знаков, правовые, перевод на поддерживаемые языки  - Контент: аудио, видео, изображения, текст, документация |
| **Интернационализация** |
| **Конфигурационное** |
| **Совместимости**  - кроссбраузерное тестирование |
| **Инсталяционное**  - установка  - обновление  - удаление |
| **Производительности**  - нагрузочное  - стрессовое  - стабильности (надежности)  - масштабируемости  - объемами  - конкурентное |
| **Тестирование на отказ и восстановление (помехоустойчивость)** |
| **Тестирование документации** |

**ТЕСТИРОВАНИЕ**

1 - **по степени важности:**

1. Дымовое (smoke) тестирование – проверка самой важной функциональности, неработоспособность которой делает бессмысленной саму идею работы приложения.
2. Тестирование Критического пути – проверка функциональности, используемой типичными пользователями в типичной повседневной деятельности.
3. Расширенное тестирование – проверка всей функциональности, заявленной в требованиях, даже той, которая низко проранжирована по степени важности.

*\*(Со слов Маршала: 2. Регрессионное тестирование 3. Тестирование сборки 4. Санитарное тестирование)*

2 – **по уровню детализации приложения:**

1. Модульное тестирование – проверяются отдельные небольшие части приложения.
2. Интеграционное тестирование – проверяется взаимодействие между несколькими частями приложения (каждая из которых, в свою очередь, проверена отдельно на стадии модульного тестирования).
3. Системное тестирование – приложение проверяется целиком как единое целое, собранное из частей проверенных на предыдущих этапах.
4. Приемочное тестирование

3 – **по доступу кода и архитектуре приложения:**

1. Черный ящик – доступ к коду есть
2. Белый ящик – доступа к коду нет
3. Серый ящик – к части код есть, к части – нет

4 – **по запуску кода на исполнение:**

1. Статическое – тестирование без запуска кода на исполнение (Документы, эскизы, параметры среды).
2. Динамическое – тестирование с запуском кода на исполнение.

5 – **По степени (привлечения конечных пользователей) удаленности:**

1. Альфа – выполняется внутри организации-разработчика с возможным привлечением конечных пользователей.
2. Бета - выполняется вне организации-разработчика с активным привлечением конечных пользователей/заказчиков.
3. Гамма – финальная стадия тестирования перед выпуском продукта, направленная на исправление незначительных дефектов, обнаруженных в бета-тестировании.

6 – **По принципам работы с приложением:**

1. Позитивное – все действия с приложением выполняются строго по инструкции, без каких бы то ни было недопустимых действий, некорректных данных и т.д.
2. Негативное – в работе с приложением выполняются (некорректные) операции и используются данные, потенциально приводящие к ошибкам.

7 – **Классификация по природе приложения:**

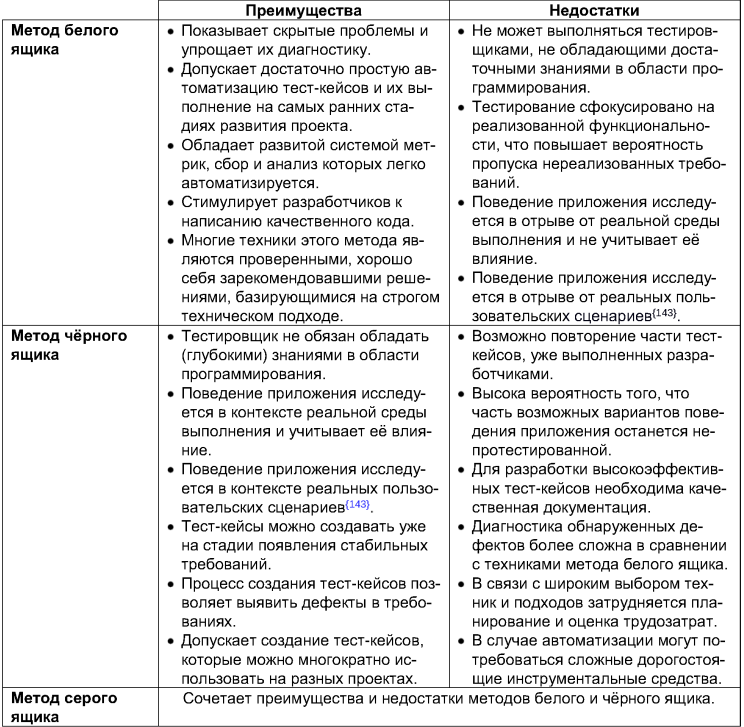
1. Тестирование веб-приложений – сопряжено с интенсивной деятельностью в области тестирования совместимости, тестирования производительности.
2. Тестирование мобильных приложений – требует повышенного внимания к тестированию совместимости, оптимизации производительности, автоматизации с применением эмуляторов мобильных устройств.
3. Тестирование настольных приложений – является самым классическим среди всех перечисленных в данной классификации, и его особенности зависят от предметной области приложения, нюансов, архитектуры, ключевых показателей качества.

8 – **Классификация по степени формализации:**

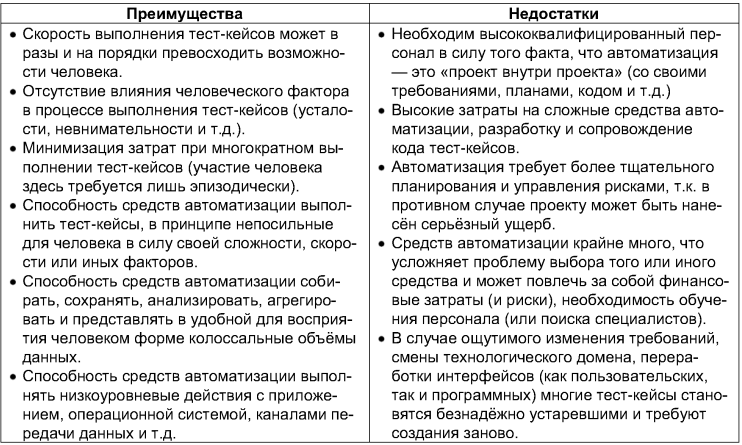
1. Тестирование на основе тест-кейсов – формализованный подход, в котором тестирование производится на основе заранее подготовленных тест-кейсов, наборов тест-кейсов и иной документации.
2. Исследовательское тестирование – частично формализованный подход, в рамках которого тестировщик выполняет работу с приложением по выбранному сценарию, который в свою очередь, дорабатывается в процессе выполнения с целью более полного исследования приложения.
3. Свободное (интуитивное) тестирование – полностью неформализованный подход, в котором не предполагается использование тест-кейсов, чек-листов, сценариев – тестировщик полностью опирается на свой профессионализм и интуицию.

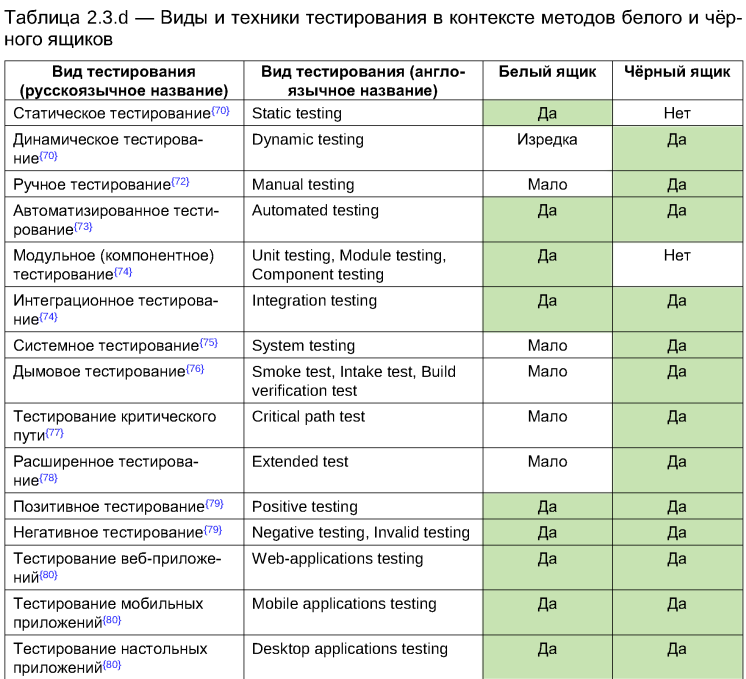
9 – **Классификация по целям и задачам:**

1. Функциональное тестирование – вид, направленный на проверку корректности работы функциональности приложения. Часто по методу черного ящика.
2. Нефункциональное тестирование – вид, направленный на проверку особенностей приложения, таких как удобство пользования, совместимость, производительность, безопасность и т.д.
3. Инсталляционное тестирование – тестирование, направленное на выявление дефектов, влияющих на протекание стадии инсталляции приложения (установка, обновление, изменение на старую версию, переустановка, удаление).
4. Регрессионное тестирование – тестирование, направленное на проверку того факта, что в ранее работоспособной функциональности не появились ошибки, вызванные изменениями в приложении или среде его функционирования.
5. Повторное тестирование – выполнение тест-кейсов, которые ранее обнаружили дефекты, с целью, подтверждения устранения дефектов.
6. Приемочное тестирование – формализованное тестирование, направленное на проверку приложения с точки зрения конечного пользователя/заказчика и вынесения решения о том, принимает ли заказчик работу у исполнителя.



**Ручное и Автоматизированное Тестирование**

****

****

**АРТЕФАКТЫ ТЕСТИРОВАНИЯ**

|  |  |
| --- | --- |
| **Чеклисты** | **Тестовый набор (test suite)** |
| Чеклист (идея проверки) | Тест-кейс (проверка, состоящая из последовательных шагов и ожидаемого результата) |

|  |  |
| --- | --- |
| **Чеклист** | |
| **АТТРИБУТЫ** |  |
| Номер |  |
| Описание проверки |  |
| Статус | Passed  Failed  Skipped  No run  Blocked |
| Комментарии |  |

|  |  |
| --- | --- |
| ПЛЮСЫ | МИНУСЫ |
| Гибкость  Простота создания и поддержки  Простота визуализации  Расширение тестового покрытия | Каждый выполняет по-своему  Неопределенность тестовых данных  Не эффективен для джунов  Высокая вероятность разночтения |

***Test Case*** - это конкретные шаги, которые нужно выполнить для проверок. (набор входных значений, предусловий, результатов, постусловий для выполнения определенного пути программы)

|  |  |
| --- | --- |
| **Тест-кейс** | |
| **АТТРИБУТЫ** |  |
| *Номер кейса* |  |
| *Заголовок* |  |
| *Предусловие* |  |
| *Шаги проверки* |  |
| *Ожидаемый результат* |  |
| Номер |  |
| Описание проверки |  |
| Статус | Passed  Failed  Skipped  No run  Blocked |
| Комментарии |  |

**ПРАВИЛА ТЕСТ-КЕЙСОВ**

1. Независимость – (Перед этим тестом пройдите 1 тест)
2. Однозначность – (ввести логин латиницей)
3. Полнота – (Никто не должен додумывать или спрашивать инфо)
4. Обезличенность – (Нажать на кнопку)
5. Упрощение – (Нажать на кнопку «Войти)
6. 1 тест-кейс – 1 цель (Авторизация и покупка товара)

**ТЕХНИКИ ТЕСТ-ДИЗАЙНА**

1. Классы эквивалентности
2. Граничные значения
3. Таблица принятия решений
4. ADHOC
5. Сценарная
6. Доменный анализ
7. Попарное тестирование
8. Предугадывание ошибок
9. Причина / Следствие
10. Состояние и переходы
11. Валидация / Верификация
12. Позитивные / Негативные
13. Исчерпывающее тестирование

**Дефект, баг** – несовершенство или недостаток в рабочем продукте, если он не соответствует его требованиям или спецификации. Недостаток в компоненте или системе, способный привести к ситуации сбоя или отказа.

**Сбой** – самоустраняющийся отказ или неоднократный отказ, устраняемый вмешательством оператора.

**Отказ** – событие, заключающееся в нарушении работоспособного состояния объекта.

**Ожидаемый результат** – поведение системы, описанное в требованиях.

**Фактический результат** – поведение системы, наблюдаемое в процессе тестирования.

**Ошибка** – действия человека, приводящие к некорректным результатам.

**Баг-репорт** – документ, описывающий ситуацию или последовательность действий, приведшую к некорректной работе объекта тестирования, с указанием причин и желаемого результата.

Баг-трекеры (Софт для заведения баг-репортов):

* Jira
* Redmine
* Mantis

**АТРИБУТЫ БАГ-РЕПОРТА**

|  |  |
| --- | --- |
| **ОСНОВНЫЕ** | **ДОПОЛНИТЕЛЬНЫЕ** |
| * Идентификатор | * Приложения – скриншоты, видео |
| * Заголовок | * Возможность обойти баг |
| * Шаги воспроизведения | * Воспроизводимость |
| * Актуальный результат | * Комментарии |
| * Ожидаемый результат |  |
| * Окружение |  |
| * Критичность |  |
| * Приоритет |  |

**Заголовок**

1. Что произошло?
2. Где произошло?
3. При каких условиях?

* Не работает поиск
* Приложение не работает
* Орфографическая ошибка

Правильно:

* нет логотипа на главной странице, если пользователь с ролью user

|-----------------| |-------------------------| |----------------------------------------|

Что Где При каких условиях

**Шаги воспроизведения**

* Описывайте каждое действие в отдельном шаге
* Описывайте безличные формулировки, призывающие к действию
* Описывайте каждый шаг, пока не столкнетесь с дефектом
* Найдите кратчайший путь воспроизведения
* Найдите точный путь воспроизведения
* Пишите так, чтобы любой новичок мог его воспроизвести

Пример:

|  |
| --- |
| Не отображается логотип на главной странице профиля  Шаги воспроизведения:   1. Перейти по ссылке: <https://example.com> 2. Ввести логин “login” 3. Ввести пароль “password” 4. Кликнуть sign in   Актуальный результат: В левом верхнем углу вместо логотипа – пусто (см. скриншот 1)  Ожидаемый результат: В верхнем левом углу находится логотип (согласно треб. 12) |

**Severity (Критичность)**

*Степень влияния дефекта на работу системы*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **01**  **Blocker** |  |  |  |  |
|  | **02**  **Critical** |  |  |  |
|  |  | **03**  **Major** |  |  |
|  |  |  | **04**  **Minor** |  |
|  |  |  |  | **05**  **Trivial** |

**01 Blocker** – блокирующая ошибка

* После этого дефекта, мы не сможем продолжать работать с системой в целом или конкретной функциональностью (ключевой)
* Для продолжения тестирования, необходимо исправление этого дефекта
* Может измениться на **critical**, если найден обходной путь

**02 Critical** – критическая ошибка

* Неправильно работающая ключевая бизнес-логика
* Дыра в системе безопасности
* Проблема приводит к временному падению сервера или приводит в нерабочее состояние часть системы

**03 Major** – значительная ошибка

* Часть основной бизнес логики работает некорректно
* Есть возможность работать с тестируемым объектом, используя другие тестовые данные
* Обычно, ставится в случаях, когда функционал может работать при одних условиях, но падает при других

**04 Minor** – незначительная ошибка

* Не нарушает бизнес-логику
* Очевидная проблема UI/UX
* Грамматические ошибки

**05 Trivial** – тривиальная ошибка

* Не затрагивает функциональность системы
* Приводит к ошибкам в сторонних программах, но не в нашей
* Опечатки, пропуски пробела
* Незначительные отклонения верстки

**Priority (Приоритет)**

*Очередность выполнения задачи или устранения дефекта. Чем выше приоритет, тем быстрее нужно исправить дефект / влияние дефекта на бизнес*

|  |  |  |
| --- | --- | --- |
| 01  High |  |  |
|  | 02  Normal |  |
|  |  | 03  Low |

**01 High** – Ошибка должна быть исправлена как можно быстрее, т.к. ее наличие является критической для проекта

**02 Normal** – Ошибка должна быть исправлена, ее наличие не является критичной, но требует обязательного решения

**03 Low** – Ошибка должна быть исправлена, ее наличие не является критичной и не требует срочного решения

**СОВЕТЫ**

* *Убирайте лишние шаги*
* *Упрощайте шаги, чем длиннее шаг/описание шага, тем хуже*
* *Проверьте на дубликаты*
* *Убедитесь, что репорт очевидный и понятный*
* *1 отчет для 1 дефекта*
* *Прослеживаемость с другими артефактами*
* *Проверьте грамотность*

**Основные ошибки:**

* Недостаточно предоставленных данных, для воспроизведения бага, или баг воспроизводится только при конкретных обстоятельствах, но они не указаны
* Название репорта и ожидаемый результат не соответствуют друг другу
* Завышение/занижение Severity
* Неверное употребление терминологии
* Сложные речевые обороты
* Отсутствует ожидаемый результат
* Критика программиста

**Когда баг не исправят:**

* Программист не смог воспроизвести баг
* Нет актуального поведения
* Нет ожидаемого поведения
* Репорт написан расплывчато/непонятно
* Нет необходимых скриншотов/логов
* QA не смог объяснить важность проблемы

**Релиз бага** – это когда программное обеспечение или приложение передается группе тестирования, зная, что дефект присутствует в выпуске. При этом приоритет и серьезность ошибки низки

**Утечка бага** – когда баг обнаруживается конечными пользователями или заказчиком, а не обнаруживается группой тестирования во время тестирования программного обеспечения

**Откуда берутся баги:**

* Недопонимание
* Ошибки программирования
* Сжатые сроки
* Изменения в требованиях
* Сложность ПО

**Отчет о тестировании**

**Отчет о тестировании** – это анализ результатов тестирования. Проводится с некоторой периодичностью в процессе работы с проектом, а также в конце работы с проектом.

**Тест-план** – документ, описывающий и регламентирующий перечень работ по тестированию, а также соответствующие техники и подходы, стратегию, области ответственности, ресурсы, расписание и ключевые даты.

**Планирование** – непрерывный процесс принятия управленческих решений и методической организации усилий по их реализации с целью обеспечения качества некоторого процесса на протяжении длительного периода времени.

**Его задача** – оценить текущее или финальное качество проекта и принять соответствующее решение и меры.

Отчет включает в себя: часть текстовой документации, описания процессов тестирования, суммарная информация о протестированных за подотчетный период билдах, информация о деятельности тестировщиков, другие статистические данные.

Цель отчета – предоставить лицам, заинтересованным в проекте, полную и объективную информацию о текущем состоянии качества проекта. Выражается в фактах и цифрах.

**Периодичность отчетов**

Отчет создается по расписанию. Например – раз в неделю. Может создаваться по выходу билдов или сроков «Спринта».

* Можно привязать к выпуску билдов
* За выпуск отвечает лидер тестировщиков
* Отчет создается на основе принятого в компании или представленного заказчиком шаблона

**Структура отчета**

|  |
| --- |
| Команда тестировщиков |
| Описание процесса тестирования (testing process description) |
| Краткое описание (summary) |
| Расписание (testing timetable) |
| Рекомендации (recommendations) |
| Статистика по ошибкам (bugs statistics) |
| Список новых ошибок (new bugs found) |
| Статистика по всем ошибкам (all bugs statistics) |

|  |
| --- |
| **ВОПРОС:** Цель тестирования? |

**ОТВЕТ:** проверка соответствия ПО предъявляемым требованиям, обеспечение уверенности в качестве ПО, поиск очевидных ошибок в программном обеспечении, которые должны быть выявлены до того, как их обнаружат пользователи программы.

|  |
| --- |
| **ВОПРОС:** Какие бывают требования? |

**ОТВЕТ:** Требования бывают **прямые** и **косвенные**, которые делятся:

|  |  |
| --- | --- |
| **Функциональные (характер)** | **Не Функциональные (характер)** |
| Бизнес требования | Производительность |
| Требования пользователей | Интерфейсы работы |
| Функциональные требования |  |

|  |
| --- |
| **ВОПРОС:** Самый приоритетный вид тестирования? |

**ОТВЕТ:** Smoke тестирование выполняется всегда в первую очередь. Все остальные наборы тестов идут уже после успешно пройденного smoke. У многих более менее крупных проектов смоки автоматизированы и запускаются не только на предрелизных билдах но и на всех промежуточных. После успешного smoke тестирования идет основная проверка функционала, соответвественно, это функциональное тестирование.

|  |
| --- |
| **ВОПРОС:** Самый низко приоритетный вид тестирования? |

**Ответ:** Все зависит от продукта и самый наименее эффективный вид тестирования для продукта будет самым низкоприоритетным

|  |
| --- |
| **ВОПРОС:** Этапы тестирования? |

**Ответ:**

1. Анализ продукта
2. Работа с требованиями
3. Разработка стратегии тестирования и планирование процедур контроля качества
4. Создание тестовой документации
5. Тестирование прототипа
6. Основное тестирование
7. Стабилизация
8. Эксплуатации

|  |
| --- |
| **ВОПРОС:** Случаи для чеклиста/тест-кейса? |

**Ответ:** Чек-лист содержит только действия, без ожидаемого результата. Чек-лист менее формализован. Общий план проверки. Тест-кейс более подробный.

Тест кейс необходим для разработки особо важных программ, например военная сфера, медицина и т.п, а чек лист в таких продуктах не подойдет.

|  |
| --- |
| **ВОПРОС:** Когда следует заканчивать тестирование? |

**Ответ:** Тестирование может быть завершено когда:

* Все 100% требований учтены.
* Дефекты установлены/ожидаемое число дефектов обнаружено.
* Все дефекты, относящиеся к классу Show Stopper или Blocker, устранены, ни у одного из критических дефектов нет статуса «открытый».
* Все дефекты с высоким приоритетом идентифицированы и исправлены.
* Defect Rate (скорость дефектообразования) ниже установленного допустимого уровня.
* Очень небольшое число дефектов среднего уровня критичности «открыты», их разбор проведен.
* Число «открытых» дефектов среднего уровня, которые не влияют на пользование системой, очень небольшое.
* Все дефекты с высоким уровнем приоритета закрыты и соответствующие регрессивные сценарии успешно проведены.

|  |
| --- |
| **ВОПРОС:** Какие причины приводят к багам? |

**Ответ:**

* Недопонимание
* Ошибки программирования
* Сжатые сроки
* Изменения в требованиях
* Сложность ПО

|  |
| --- |
| **ВОПРОС:** Приведите пример End-to-end теста? |

**Ответ:** В данном случае end-to-end будет означать, что тест начался у конечного пользователя, который инициировал сценарий и закончился у него же, либо, начался у одного пользователя, и закончился у другого (например, в начале - пользователь-абонент ( первый end), сценарий "пополнение счета", в конце - система биллинга, которая засчитывает все изменения на счету (второй end)).

**ВОПРОС:** В каком случае тестирование может доказать отсутствие дефектов?

**ВОПРОС:** Какие ошибки допущены в требовании: «Игрок должен иметь возможность погрузится в воду, пока не выйдет время»?

|  |
| --- |
| **ВОПРОС:** Чем Баг-репорт отличается от Тест-кейса? |

**Ответ:** **Баг-репорт** – документ, описывающий ситуацию или последовательность действий, приведшую к некорректной работе объекта тестирования, с указанием причин и желаемого результата.

***Test Case*** - это конкретные шаги, которые нужно выполнить для проверок. (набор входных значений, предусловий, результатов, постусловий для выполнения определенного пути программы)

**ВАЖНО!**

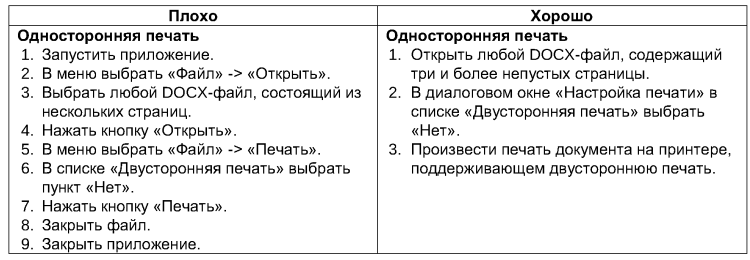
**Front end** – это непосредственный интерфейс пользователя, т.е. текст, картинки, кнопки, ссылки и прочие вещи, которые пользователь видит в окне веб-браузера или e-mail клиента (других программ).

**Back end** – это ПО и данные, находящиеся за фасадом Front end: HTML-код веб-страницы, веб-сервер, код приложения, база данных и т.д.

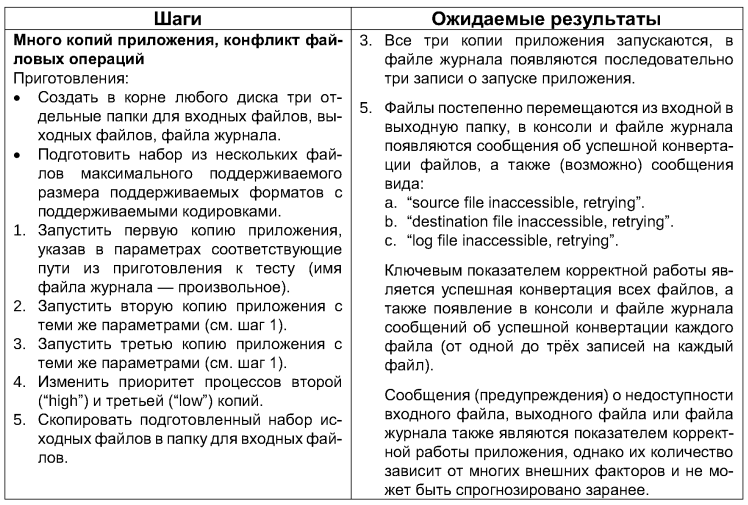
**Доменное тестирование** – техника создания эффективных и результативных тест-кейсов в случае, когда несколько переменных могут или должны быть протестированы одновременно.

**Попарное тестирование** – техника тестирования, в которой вместо проверки всех возможных комбинаций значений всех параметров проверяются только комбинации значений каждой пары параметров.

**Так выглядит нормальный тест-кейс**



**Хороший сложный тест-кейс**



**Так выглядит нормальный чек-лист**



**Верификация** — статический процесс соответствия установленным в спецификациях требованиям (проверка наличия).

**Валидация** — динамический процесс соответствия ожиданиям и требованиям клиента (проверка работоспособности).

## СТАДИИ ЖИЗНЕННОГО ЦИКЛА ОШИБКИ

1. Тестировщик обнаруживает дефект.

2. Тестировщик пишет отчет об ошибке в систему управления дефектами (статус **New (новый)**) и перенаправляет его на разработчика (статус **Assigned (назначен)**).

3. Разработчик изучает ошибку, ее возможности воспроизведения и по полученным результатам соотносит ее к одному из статусов:

* **Duplicate (дубликат)** – подобный дефект уже существует в системе по отслеживанию ошибок;
* **Rejected (отклонен)** – ошибка не требует внесения корректив, поскольку ее влияние на продукт незначительное;
* **Deferred (отсрочен)** – корректировку данной ошибки можно осуществить в другой версии программы;
* **Not a bug (не баг)** – дефект не есть ошибкой, поэтому вносит коррективы не требуется;
* **Open (открыт)** – дефект в процессе исправления;
* **Fixed (исправлен)** – код изменен и протестирован разработчиком.

4.Тестировщик повторно проверяет ошибку (статус «**Retesting» (повторное тестирование)**).

5. Если дефект исправлен, тестировщик его закрывает (статусы «**Verified»** **(проверен),** затем «**Closed»** **(закрыт)**).

6. Если дефект проявляется и дальше, он опять передается на редактирование разработчику (статусы «**Reopened»** **(переоткрыт)**, «**Assigned» (назначен))** и вновь проходит через каждую стадию цикла.

На основе статусов ошибок в системе управления дефектами, можно создавать отчеты, по которым уже оценивать работу как программиста, так и тестировщика при выполнении им возложенных [услуг по тестированию программного обеспечения.](https://testmatick.com/ru/our-qa-services/)