Вопросы к экзамену

1. Тестирование – процесс анализа ПС и сопутствующей документации с целью выявления дефектов и повышения качества продуктов (проще говоря, тестирование — это поиск багов).

Тестированию могут подвергаться программы, код (без запуска и исполнения), прототип программного продукта, проектная документация (требования, спецификации, архитектура и дизайн, тест-кейсы и сценарии), а так же сопроводительная документация.

2. Дефект – расхождение ожидаемого и фактического результата,
Ожидаемый результат – «Правильный» результат поведения,
Тест-кейс – это профессиональная документация тестировщика, последовательность действий направленная на проверку какого-либо функционала, описывающая как придти к фактическому результату,
Тест – план - это документ описывающий весь объем работ по тестированию, начиная с описания объекта, стратегии, расписания, критериев начала и окончания тестирования, до необходимого в процессе работы оборудования, специальных знаний, а также оценки рисков с вариантами их разрешения,
Билд – это «сборка» исполняемых файлов, исходников и конфигов.

3. Статическое тестирование производится без запуска программного кода продукта. Тестирование осуществляется путем анализа программного кода (code review) или скомпилированного кода. Анализ может производиться как вручную, так и с помощью специальных инструментальных средств. Целью анализа является раннее выявление ошибок и потенциальных проблем в продукте.
В отличии от статического, динамическое тестирование производится путем запуска продукта и проверки его функционала. Проверка осуществляется с помощью ручного или автоматического выполнения заранее подготовленного набора тестов.

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

5. Метод белого ящика – метод тестирования программного обеспечения, который предполагает, что внутренняя структура/устройство/реализация системы известны тестировщику.
Метод черного ящика – метод тестирования при котором не используются знания о внутреннем устройстве объекта тестирования.
Метод серого ящика – это комбинирование техник из бело и черноящичного тестирования.

6. Тестирование компонентов — тестируется минимально возможный для тестирования компонент, например, отдельный класс или функция. Часто

тестирование компонентов осуществляется разработчиками программного обеспечения.

Интеграционное тестирование — тестируются интерфейсы между компонентами, подсистемами или системами. При наличии резерва времени на данной стадии тестирование ведётся итерационно, с постепенным подключением последующих подсистем.

Системное тестирование — тестируется интегрированная система на её соответствие требованиям.

7. Функциональное тестирование — это тестирование ПО в целях проверки реализуемости функциональных требований, то есть способности ПО в определённых условиях решать задачи, нужные пользователям. Функциональные требования определяют, что именно делает ПО, какие задачи оно решает.

Функциональные требования включают в себя:

Функциональная пригодность (англ. suitability).

Точность (англ. accuracy).

Способность к взаимодействию (англ. interoperability).

Соответствие стандартам и правилам (англ. compliance).

Защищённость (англ. security).

8. Качество программного обеспечения (Software Quality) — это совокупность характеристик программного обеспечения, относящихся к его способности удовлетворять установленные и предполагаемые потребности.

качество тестируемой программы наиболее рационально определять, ссылаясь на следующие маркеры:

преимущества, благодаря которым она понравится заказчику и потенциальным потребителям;

недостатки, наличие которых обусловит ситуацию отказа клиента от этой программы и приобретение другой.

полное выполнение требований заданных заказчиком в спецификации

9. Функциональные виды тестирования

• Функциональное тестирование (Functional testing)
• Тестирование безопасности (Security and Access Control Testing)
• Тестирование взаимодействия (Interoperability Testing)

Нефункциональные виды тестирования

• Все виды тестирования производительности:
o нагрузочное тестирование (Performance and Load Testing)
o стрессовое тестирование (Stress Testing)
o тестирование стабильности или надежности (Stability / Reliability Testing)
o объемное тестирование (Volume Testing)
• Тестирование установки (Installation testing)
• Тестирование удобства пользования (Usability Testing)
• Тестирование на отказ и восстановление (Failover and Recovery Testing)
• Конфигурационное тестирование (Configuration Testing)

Связанные с изменениями виды тестирования

• Дымовое тестирование (Smoke Testing)
• Регрессионное тестирование (Regression Testing)

• Повторное тестирование (Re-testing)
• Тестирование сборки (Build Verification Test)
• Санитарное тестирование или проверка согласованности/исправности (Sanity Testing)

10. Виды требований по уровням

Требования к программному обеспечению — совокупность утверждений относительно атрибутов, свойств или качеств программной системы, подлежащей реализации. Создаются в процессе разработки требований к программному обеспечению, в результате анализа требований.

Требования могут выражаться в виде текстовых утверждений и графических моделей.

Бизнес-требования — определяют назначение ПО, описываются в документе о видении (vision) и границах проекта (scope).

Пользовательские требования — определяют набор пользовательских задач, которые должна решать программа, а также способы (сценарии) их решения в системе. Пользовательские требования могут выражаться в виде фраз утверждений, в виде сценариев использования (англ. use case), пользовательских историй (англ. user stories), сценариев взаимодействия (scenario).

Функциональные требования — охватывают предполагаемое поведение системы, определяя действия, которые система способна выполнять[источник не указан 2073 дня]. Описывается в системной спецификации (англ. system requirement specification, SRS).

11. Пути выявления требований:

Интервью, опросы, анкетирование
Мозговой штурм, семинар
Наблюдение за производственной деятельностью, «фотографирование» рабочего дня
Анализ нормативной документации
Анализ моделей деятельности
Анализ конкурентных продуктов
Анализ статистики использования предыдущих версий системы

12. https://github.com/maxvipon/IEEE-Std-830-1998-RU/blob/master/IEEE%20STD%20830-1998%20(RU).md#5

SRS должна быть:

1. корректной;
2. непротиворечивой;
3. полной;
4. целостной;
5. упорядоченной по важности и/или стабильности;
6. верифицируемой;

7. модифицируемой;
8. трассируемой.

13. См выше

14. Основные проблемы управления требованиями, с которыми приходится сталкиваться при их анализе, сводятся к следующему.

- Требования имеют много источников.
- Требования не всегда очевидны.
- Требования не всегда легко выразить словами.
- Существует множество различных типов требований и различных уровней их детализации.
- Требования почти всегда взаимосвязаны и взаимозависимы, часто противоречивы.
- Требования всегда уникальны.
- Набор требований чаще всего является компромиссом.
- Требования изменяются.
- Требования зависят от времени.
- Требования очень трудно оценивать.

15. ??

16. Классы эквивалентности – это такое множество значений, каждое из которых подчиняется одной и той же логике в коде. Т.е. взяв хотя бы одно из них и протестировав можно сказать что тест будет идентичен для других из этого класса.

Граничные условия – это значения, лежащие на границе класса эквивалентности (верхней и нижней).

17. К основным атрибутам тест-кейса можно отнести:

- ID
- Приоритет
- Связанное требование (ссылка на требование)
- Модуль и подмодуль приложения
- Заголовок (суть тест-кейса)
- Дата создания и кем был создан
- Поле с описанием исходных данных и шагами для выполнения теста
- Ожидаемый результат по каждому шагу тест-кейса

18. Свойства качественный тест-кейсов:

- Правильный технический язык (исп. Безличную форму глагола, лаконичное описание, точные имена и названия элементов и т.д.)
- Баланс между специфичностью и общностью
- Баланс между простотой и сложностью
- «Показательность» (высокая вероятность обнаружения ошибки)
- Последовательность в достижении цели (когда все действия в тест-кейсе направлены на следование единой логике и достижении единой цели и не содержат никаких отклонений)
- Не избыточность по отношению к другим тест-кейсам
- Отсутствие лишних действий

- Демонстративность (способность демонстрировать обнаруженную ошибку очевидным образом)
- Прослеживаемость
- Возможность повторного использования
- Соответствия принятым шаблонам оформления и традициям.

19. Она сказала что этого вопроса нету в чистом виде но про тестовые сценарии может быть вопрос связанный с тест кейсом.

20. Дефект– расхождение фактического результата от ожидаемого.

Ошибка – действие человека, приводящее к некорректным результатам.

Отчет об ошибке – документ описывающий и приоритизирующий обнаруженный дефект, а так же содействующий его устранению.

Цели написания:

- Предоставить информацию о проблеме
- Приоритизировать проблему
- Содействовать устранению проблемы

Атрибуты:

- Идентификатор
- Краткое описание
- Подробное описание
- Шаги по воспроизведению
- Воспроизводимость
- Важность
- Срочность
- Симптом
- Возможность обойти
- Комментарии
- Приложение

21. Жизненный цикл бага

Обнаружен – Назначен – Исправлен – Проверен – Закрыт\Открыт заново

Так же еще может быть Статус: Отклонен (если найденное нечто является фичей например) или Отложен (если нет возможности протестировать его на данном этапе).

22. Свойства хороших отчетов об ошибках:

- Тщательное заполнение всех полей точной и корректной информацией
- Правильный технический язык
- Специфичность описания шагов (в отличие от тест-кейсов, т.к. в воспроизведении дефекта может быть важна каждая мелка деталь)
- Отсутствие лишних действий и\или длинных описаний
- Отсутствие дубликатов
- Очевидность и понятность
- Прослеживаемость
- Для каждого нового дефекта новый отчет
- Соответствие принятым шаблонам и традициям

23. Особенности тестирования веб-приложений

Технологические отличия.

Классическое приложение работает с использованием одной или семейства родственных технологий.

Web-приложение работает с использованием принципиально различных технологий.

Структурные отличия.
Классическое приложение “монолитное”. Состоит из одного или небольшого количества модулей. Не использует серверы БД, web-серверы и т.д.

Web-приложение — “многокомпонентное”. Состоит из большого числа модулей. Обязательно использует серверы БД, web-серверы, серверы приложений.

Отличия режимов работы.
Классическое приложение работает в режиме реального времени, т.е. известно о действиях пользователя сразу же, как только оно выполнено.

Web-приложение работает в режиме “запрос-ответ”, т.е. известно о некотором наборе действий только после запроса на сервер.

Особенности тестирования web-приложений, режим работы

Отличия формирования интерфейса.
Классическое приложение использует для формирования интерфейса пользователя относительно устоявшиеся и стандартизированные технологии.

Web-приложение использует для формирования пользовательского интерфейса стремительно развивающиеся технологии, множество которых конкурирует между собой.

Отличия работы с сетью.
Классическое приложение практически не использует сетевые каналы передачи данных.

Web-приложение активно использует сетевые каналы передачи данных.

Отличия запуска и остановки.
Классическое приложение запускается и останавливается редко.

Web-приложение запускается и останавливается по факту поступления каждого запроса, т.е. очень часто.

Разница в количестве пользователей.

Классическое приложение: количество пользователей, одновременно использующих приложение, подвержено контролю, ограничено и легко прогнозируемо.

Web-приложение: количество пользователей, одновременно использующих приложение, сложнопрогнозируемо и может скачкообразно меняться в широких диапазонах.

Особенности сбоев и отказов.
Классическое приложение: выход из строя тех или иных компонентов сразу становится очевидным.

Web-приложение: выход из строя некоторых компонентов оказывает непредсказуемое влияние на работоспособность приложения в целом.

Отличия в инсталляции.
Классическое приложение — процесс инсталляции стандартизирован и максимально ориентирован на широкую аудиторию пользователей. Не требует специфических знаний. Добавление компонентов приложения выполняется стандартным способом с использованием одного и того же инсталлятора.

Web-приложение — процесс инсталляции часто недоступен конечному пользователю. Инсталляция требует специфических знаний. Процесс изменения компонент приложения не предусматривается или требует квалификации пользователей. инсталлятор отсутствует.

Отличия в деинсталляции.
Классическое приложение: процесс деинсталляции стандартизирован и выполняется автоматически или полуавтоматически.

Web-приложение: процесс деинсталляции требует специфических знаний для вмешательства администратора и часто сопряжен с изменением кода среды функционирования приложения, БД, настройки системного ОС.

Особенности среды функционирования.
Классическое приложение: среда функционирования стандартизирована и не сильно влияет на функционирование приложения.

Web-приложение: среда функционирования очень разнообразна и может оказать серьезное влияние на работоспособность и серверной, и клиентской части.

24. Инсталляционное тестирование (installation testing, installability testing165) — тестирование, направленное на выявление дефектов, влияющих на протекание стадии инсталляции (установки) приложения. В общем случае такое тестирование проверяет множество сценариев и аспектов работы инсталлятора в таких ситуациях, как: о новая среда исполнения, в которой приложение ранее не было инстал- лировано; о обновление существующей

версии («апгрейд»); о изменение текущей версии на более старую («даунгрейд»); о повторная установка приложения с целью устранения возникших про- блем («переинсталляция»); о повторный запуск инсталляции после ошибки, приведшей к невозмож- ности продолжения инсталляции; о удаление приложения; о установка нового приложения из семейства приложений; о автоматическая инсталляция без участия пользователя.

25. Являясь обычной программой, инсталлятор обладает рядом особенностей, среди которых стоит отметить следующие:

● Глубокое взаимодействие с операционной системой и зависимость от неё (файловая система, реестр, сервисы и библиотеки).

● Совместимость как родных, так и сторонних библиотек, компонентов или драйверов, с разными платформами.

● Удобство использования: интуитивно понятный интерфейс, навигация, сообщения и подсказки.

● Дизайн и стиль инсталляционного приложения.

● Совместимость пользовательских настроек и документов в разных версиях приложения.

● И многое другое.

Распишем подробнее, "Что?" необходимо проверить, для оценки правильности работы инсталлятора:

● Установка (Инсталляция).

● Корректность списка файлов в инсталляционном пакете:
при выборе различных типов установки, либо установочных параметров список файлов и пути к ним также могут отличаться.
отсутствие лишних файлов (проектные файлы, не включенные в инсталляционный пакет, не должны попасть на диск пользователя).

● Регистрация приложения в ОС.

● Регистрация расширений для работы с файлами:
для новых расширений.
для уже существующих расширений.

● Права доступа пользователя, который ставит приложение:
права на работу с системным реестром.
права на доступ к файлам и папкам, например *%Windir%\system32*.

● Корректность работы мастера установки (Installation Wizard).

● Инсталляция нескольких приложений за одни заход.

● Установка одного и того же приложения в разные рабочие директории одной рабочей станции.

● Обновление.

● Правильность списка файлов, а так же отсутствие лишних файлов:
проверка списка файлов при разных параметрах установки.
отсутствие лишних файлов.

● Обратная совместимость создаваемых данных:
сохранность и корректная работа созданных до обновления данных.
возможность корректной работы старых версий приложения с данными, созданными в новых версиях.

● Обновление при запущенном приложении.

● Прерывание обновления.

● Удаление (Деинсталляция).

● Корректное удаление приложения:
удаление из системного реестра установленных в процессе инсталляции библиотек и служебных записей.
удаление физических файлов приложения.
удаление/восстановление предыдущих файловых ассоциаций.
сохранность файлов созданных за время работы с приложением.
удаление при запущенном приложении.
удаление с ограниченным доступом к папке приложения.
удаление пользователем без соответствующих прав.

31. Инструментальные средства тестирования веб-ориентированных приложений по методу белого ящика: примеры ПО и области его применения

SoapUI — это инструмент для кросс-платформенного тестирования с открытым исходным кодом. Он может автоматизировать функциональные, регрессионные, согласованные и нагрузочные тесты как SOAP, так и REST-сервисов. Он прост в использовании и поддерживает передовые технологии и стандарты для моделирования и стимулирования поведения веб-сервисов.

SOAPSonar обеспечивает комплексное тестирование веб-сервисов для HTML, XML, SOAP, REST и JSON. Он обеспечивает функциональное тестирование , тестирование производительности, совместимости и тестирование безопасности с помощью стандартов OASIS и W3C.

WebInject — это бесплатный инструмент для автоматизированного функционального, приемного и регрессионного тестирования веб-сервисов.

32 Валидация HTML/CSS-кода: определение, основные направления, преимущества и недостатки

Валидация представляет собой проверку документа специальной программой - валидатором на соответствие установленным веб-стандартам и обнаружение имеющихся погрешностей. Данные стандарты названы спецификацией (разрабатывается Консорциумом Всемирной паутины, или сокращенно W3C)

Валидацией называется проверка CSS-кода на соответствие спецификации CSS2.1 или CSS3. Соответственно, корректный код, не содержащий ошибок, называется валидный, а не удовлетворяющий спецификации — невалидный

Проверка происходит по 4-ём основным пунктам:

1. Валидация синтаксиса — проверка кода на наличие синтаксических ошибок в нем.

2. Проверка вложенности тэгов — проверка закрытия тегов, в отношении к их открытию.

3. Валидация DTD — проверяет соответствует ли код сайта указанному типу документа DTD.

4. Посторонние элементы — проверка на присутствие в коде сторонних элементов, которые не соответствуют объявленному DTD.

Плюсы валидации

Хотя HTML-код имеет достаточно простую иерархическую структуру, при разрастании объема документа в коде легко запутаться, следовательно, просто и совершить ошибку. Браузеры, несмотря на явно неверный код, в любом случае постараются отобразить веб-страницу. Но поскольку не существует единого регламента о том, как же должен быть показан «кривой» документ, каждый браузер пытается сделать это по-своему. А это в свою очередь приводит к тому, что один и тот же документ может выглядеть по-

разному в популярных браузерах. Исправление явных промахов и систематизация кода приводит, как правило, к стабильному результату.

Минусы валидации.

Сайты, конечно же, делают для того, чтобы их посещали люди. Именно посетители выступают мерилом работы сайта, а их интересует информация и способ ее получения. Пользователь желает, чтобы сайт корректно отображался в его любимом браузере, быстро загружался и содержал те материалы, которые ему нужны. Заметьте, в этом списке нет ничего про код документа и его валидность, посетителей это просто не интересует. Поэтому совершенно невалидный сайт, но выполненный с душой, наполненный интересными материалами привлечет к себе больше посетителей, чем пустой ресурс, но сделанный по всем «правилам».

Средства для проверки валидации html Markup Validation Service W3C, Web Page Analyzer, Browsershots

33. Способы тестирования ссылок в веб-ориентированных приложениях.

Битые ссылки бывают 2х типов — внутренние и внешние. Внутренние — ведущие на страницы в пределах существующего сайта. Внешние — ведущие на другие веб-ресурсы.

Программы проверок битых ссылок:

Screaming Frog SEO Spider Tool, Netpeak Spider, Xenu’s Link Sleuth

34. Виды веб-форм, особенности тестирования форм каждого вида.

Формы - это фрагменты HTML-документов, "ответственные" за ввод информации клиентом.

формы размещаются в контейнере <form>...</form>.

И используются теги

* Текстовое поле (text field): <input type="Text" ...

* Текстовая область (text area): <textarea name="имя" cols="число столбцов" ...

* Список: <option ..

* "Флажок" (Checkbox) <input type=" Checkbox" ..

* "Радиокнопка" (Radiobutton) <input type=" Radiobutton " ...

Итак, формы в общем случае делятся на два основных вида:

* однооконные формы – полностью (со всеми своими полями) расположены на одной странице;

* пошаговые формы – новые поля появляются (после загрузки новой страницы или в рамках старой с использованием JavaScript/AJAX) по мере заполнения уже показанных.

Переходим к тестированию.

Как обычно – краткий, "максимально универсализированный" чек-лист:

1. Расположение формы на экране перед заполнением, после неверного заполнения, после верного заполнения.

2. Сохранение/изменение URL при отправке данных из формы.

3. Отправка формы с незаполненными обязательными полями.

4. Отправка формы с неверно заполненными полями (числа вне диапазонов, строки превышают допустимую длину и т.п.)

5. Отправка формы с "нелогичными данными" (например, дата рождения – в будущем).

6. Отправка формы с полями, содержащими спецсимволы.

7. Восстановление значений полей после отправки формы с неверно заполненными полями.

8. Информативность сообщений об ошибках.

9. Функциональная сгруппированность полей формы. Информативность надписей, подсказок.

10. Работоспособность и юзабилити формы с отключённым JavaScript.

11. + использовать чек-листы по каждому стандартному полю (текстовое, числовое, дата и т.п.)

35. Основные направления тестирования однооконных форм.

* однооконные формы – полностью (со всеми своими полями) расположены на одной странице;

1. Путь (URL).

GOOD: После обработки данных формы пользователь должен попадать на некую "информативную страницу", т.е., например: пользователь хочет изменить количество заказанных товаров в корзине, он меняет число в соответствующем поле, кликает "Пересчитать", и должен оказаться снова на странице корзины. Проще всего достичь этого эффекта можно так: указать в атрибуте action тега form URL самой же страницы, на которой расположена форма.

BAD: Вместо страницы корзины появляется надпись "Спасибо, операция выполнена успешно". В лучшем случае после этого происходит редирект на страницу корзины. В худшем – пользователю приходится нажимать в браузере Back и… видеть подхваченную из кэша страницу, на которой остались старые данные.

Вывод: работа с веб-приложением должна быть похожа на работу с "обычным настольным приложением" в плане своей последовательности и плавности. Следует избегать лишних сообщений, действий и т.п. Это следует продумывать и при программировании, и при тестировании.

2. Месторасположение.

GOOD: Если форма расположена достаточно низко, нужно сделать автоматическую прокрутку к форме. Например, так: action="/order/#form" (при обработке данных) или href="/order/#form" (при изначальном переходе к форме). Перед самой формой пишем

BAD: Страница с формой или результатами выполнения операции открывается в состоянии "вверху окна браузера – верх страницы". Форма или некоторая информация о выполненной операции видна частично или не видна вообще. Даже если пользователь догадается проскроллить страницу, это всё равно неудобно. А пользователь может и не догадаться.

Вывод: для перехода к некоторой операции или следующему шагу некоторой операции должно быть достаточно выполнения ОДНОГО действия. Лишние клики, скроллинг и т.п. – всё это вредит как минимум юзабилити.

3. Ошибочные ситуации.

GOOD: В случае, если пользователь ввёл некоторые данные некорректно (или не ввёл вообще), форма должна быть показана заново, и при этом: а) все введённые данные (за исключением паролей и полей CAPTCHA) должны сохранять свои значения (ВСЕ ПОЛЯ! Т.е. и чек-боксы, и радиобаттоны, и списки и т.д.); б) в удобном для восприятия месте (чаще всего – сразу над формой) следует чётко указать причину неудачи отправки данных; все неверно заполненные поля следует визуально выделить, а рядом с полем (при возможности) указать суть ошибки заполнения и подсказку по правильному заполнению.

BAD: Иногда можно увидеть такое: а) Значения полей "обнуляются" (поля или очищаются вообще или принимают некоторые значения по умолчанию. Особенно круто, когда так себя ведёт лишь часть полей. б) Причина неудачи отправки данных не объяснена вообще или объяснена в стиле "Вы неверно заполнили некоторые поля". Развиваем интуицию пользователя? в) Ошибочно заполненные поля не указаны, суть ошибки не подчёркнута или подчёркнута способом, допускающим недопонимание. Например: "Неверно указана дата". Какая и почему? Может быть, пользователь допустил ошибку в самой дате (42 декабря) в поле "Дата рождения", а может – указал просто заведомо абсурдную дату (например: дату своего рождения как 01.01.2150).

Вывод: пользователь должен быть избавлен от необходимости быть телепатом, чтобы пользоваться вашим сайтом. Он также должен быть избавлен от необходимости повторно выполнять те действия, без повторного выполнения которых можно обойтись.

4. Поля, их значения и иже с ними.

Только что (в пункте 3) мы говорили об обработке ситуации "юзер что-то ввёл не так". Но, допустим, юзер ещё ничего не ввёл. Итак.

GOOD: Одно поле – одно значение. Согласитесь, что заполнять, обрабатывать и тестировать такое – просто (пока опустим вопросы выравнивания, цветового и графического оформления и т.д.).

BAD: А вот такое и заполнять, и обрабатывать (парсить значения) и тестировать сложно.

GOOD: Если это возможно (и, особенно, если желательно!) – помещайте рядом с полем пример его заполнения (диапазон дат, пример имени пользователя и т.п.) Это облегчает жизнь и посетителям сайта (проще заполнять форму), и программистам (проще понять, какие будут значения, какие на них можно наложить ограничения и т.д.), и тестировщикам (в случае с диапазонами пример заполнения даёт готовые граничные условия для классов эквивалентности, в случае "просто примера" сразу есть входные данные для простого позитивного теста).

BAD: Иногда из названия поля невозможно догадаться, что и в каком формате следует ввести. И если у программиста и тестировщика есть спецификации (да в них лазить, как известно, часто – лень), то у пользователя остаётся только телепатия. Пример: поле "Идентификатор товара". Ага. И какой он? Может, "Т001-24-н-87-Б"? Или "123-6"? Или "Мониторчик синенький"?

ТОО BAD! Но есть одно исключение. Категорически не рекомендуется приводить примеры паролей (по той простой причине, что львиная доля "пользователей невдумчивых" именно такой пароль и использует). Даже сгенерированный уникальный пароль лучше не приводить (он будет закэширован как часть текста страницы).

Вывод: уделяйте внимание тому, как организованы поля формы. Это, в конечном итоге, экономит время всем разработчикам и пользователям проекта.

5. Значения полей и спецсимволы.

GOOD: Следует чётко отдавать себе отчёт в том, что через поле, теоретически, могут быть введены любые символы. ЛЮ-БЫ-Е. Из этого, конечно, следует необходимость тщательной фильтрации данных перед передачей их в БД или иной приёмник. Но также следует фильтровать данные и правильно оформлять формы, имея в виду, что в случае неверного заполнения части полей данные будут снова показаны в форме (см. пункт 3).

Базовые идеи: 1. Помним и используем простое правило: значения атрибутов тегов берутся в кавычки. Без исключений. 2. Используем функции, позволяющие отобразить данные как часть HTML-кода (например, банальную функцию htmlspecialchars() в PHP).

BAD: Плохого по этому пункту можно написать много. Но самые "весёлые" ситуации возникают, когда нарушены вышеназванные "базовые идеи". Например, пользователь заполняет поле "Место проживания" так:

Отель "Минск", 2-й этаж.

Форма сделана бездарно, фильтрации нет, а в коде прописано так:

<input type=text name=place value={value} />

Когда плейсхолдер {value} будет заменён на реальное значение, код примет вид:

< … value=Отель "Минск", 2-й этаж />

В переводе на русский:

value="Отель"

Всё. Данные о названии отеля и этаже утеряны.

Вывод: переоценить правильное оформление HTML-кода и вдумчивую фильтрацию полученных от пользователя данных едва ли возможно.

6. JavaScript в формах – "за" и "против".

GOOD: JavaScript, AJAX и иже с ними – прекрасные инструменты, которые позволяют значительно "оживить" в плане эстетического восприятии, а также ускорить работу с формами. Так, например, можно проверять значения полей на корректность ещё в процессе заполнения и/или перед отправкой данных из формы. Можно выводить красивые подсказки, элегантно сообщать об ошибочных ситуациях и способах их решения.

BAD: Но! JavaScript может быть отключён. Он также может "сглючить" в силу множества сложнопредсказуемых причин. И что тогда? А тогда форма, рассчитанная на активное

использование JavaScript и лишённая альтернативных способов "общения с пользователем" становится как минимум неинформативной (принимает "bad вид" из пункта 4), а как максимум – неработоспособной.

Вывод: НЕЛЬЗЯ полагаться ТОЛЬКО на JavaScript при работе с формами. Формы и механизмы их обработки должны оставаться полностью функциональными (и терять минимум юзабилити) при отключённом JavaScript.

36. Основные направления тестирования пошаговых форм.

* пошаговые формы – новые поля появляются (после загрузки новой страницы или в рамках старой с использованием JavaScript/AJAX) по мере заполнения уже показанных.

*

Основная беда пошаговых форм – неочевидность их функционирования при возврате на предыдущие шаги или восстановлении работы после обрыва связи. Пункты 1-6 в полной мере относятся к каждому отдельному шагу пошаговой формы, а специфика – вот она:

GOOD: Следует однозначно давать понять пользователю, разрешено ли ему возвращаться на предыдущие шаги. Если разрешено – этот механизм должен быть тщательно продуман с целью исключения возможности искажения и потери данных. Если запрещено – должен быть механизм, явно запрещающий возврат, а в случае попыток совершить таковой должно появляться сообщение об ошибке. Также, возможно, следует предпринять дальнейшие шаги (например, перенаправить пользователя на первый шаг формы для повторного заполнения).

BAD: Механизм управления шагами работает "как повезёт". В зависимости от браузера и иных факторов возврат то не работает ("page has expired"), то работает нестабильно (данные искажаются и теряются), то вытворяет иные трюки и фокусы.

Вывод: пошаговые формы сложнее однооконных, а потому их разработке и тестированию следует уделять повышенное внимание.

37. Чек-лист для тестирования формы: пример.

Чек-лист - это документ, описывающий что должно быть протестировано. При этом чек-лист может быть абсолютно разного уровня детализации. На сколько детальным будет чек-лист зависит от требований к отчётности, уровня знания продукта сотрудниками и сложности продукта.

Чек лист это документ как правило с набросками идей для тестирования или улучшения, из которого как правило берут нормальные мысли и оформляют уже как тест-кейс

(Чек-лист по тестированию формы регистрации)
http://checklists.expert/checklist/1489-chek-list-po-testirovaniyu-formy-registracii

38. Основные проверки текстовых полей форм.

Текстовые поля являются элементами графического интерфейса (GUI)

GUI (Graphical User Interface) Testing — тестирование графического пользовательского интерфейса. Графический пользовательский интерфейс — это интерфейс, в котором пользователь взаимодействует с компьютером, используя графические изображения .

Общие проверки:

* Вид элементов при уменьшении окна браузера + появление скрола

* Правильность написания текста + текст должен быть выровнен

* Правильность перемещения фокуса в окне (Tab / Tab+Shift)

* Выбранные элементы выделяются

* Неизменяемые поля выглядят одинаково и отличаются от редактируемых

* Желательно не использовать двойной клик

* Проверка наличия нужных нотификейшенов

* Унификация дизайна (цвет, шрифт, размер)

* При необходимости должны быть тултипы

* Изменение вида элемента при ховере на него

* Если формы дублируются, то должны быть одинаковые названия

Основные моменты при проверке GUI:

* расположение, размер, цвет, ширина, длина элементов; возможность ввода букв или цифр

* реализуется ли функционал приложения с помощью графических элементов

* размещение всех сообщений об ошибках, уведомлением (а также шрифт, цвет, размер, расположение и орфография текста)

* читабелен ли использованный шрифт

* переходит ли курсор из текстового в поинтер при наведении на активные элементы, выделяются ли выбранные элементы

* выравнивание текста и форм

* качество изображений

* проверить расположение и отображение всех элементов при различных разрешениях экрана, а также при изменении размера окна браузера (проверить, появляется ли скролл)

* проверить текст на орфографические, пунктуационные ошибки

* появляются ли тултипы (если есть необходимость)

* унификация дизайна (цвета, шрифты, текст сообщений, названия кнопок и т.д.)

Текстовое поле

* Проверить выделение текста с помощью Ctrl+A / Shift+стрелка

* Проверка ввода длинного текста

И т.д.

39. Основные проверки чек-боксов, радио-баттонов, списков.

Для радио-баттоны, чек-боксов и списки тоже являются частью GUI и правила проверки для них тоже схожи, так же стоит учесть особенности использования каждого компонента. Смотри вопрос 38.

Дополнительные проверки для веб-форм

Чек-боксы

Установить чекбокс кликом и пробелом

Расположение возле соответствующего текста

Выпадающие списки

Должна быть функция прокрутки

Должны располагаться по алфавиту (если текст), по возрастанию (если числовые значения)

Если элемент был выбран, то должен находиться сверху либо обозначен, что он выбран

Чек-боксы

Установить чекбокс кликом и пробелом

Расположение возле соответствующего текста

40. Тестирование совместимости: определение, основные направления.

Тестирование совместимости (англ. compatibility testing) — вид нефункционального тестирования, основной целью которого является

проверка корректной работы продукта в определенном окружении. Окружение может включать в себя следующие элементы:

Аппаратная платформа;

Сетевые устройства;

Периферия (принтеры, CD/DVD-приводы, веб-камеры и пр.);

Операционная система (Unix, Windows, MacOS, ...)

Базы данных (Oracle, MS SQL, MySQL, ...)

Системное программное обеспечение (веб-сервер, файрволл, антивирус, ...)

Браузеры (Internet Explorer, Firefox, Opera, Chrome, Safari