



IT-Academy

Образовательный центр
программирования и
высоких технологий



Евгений Шидловский

Ирина Шуляк

под редакцией Андрея Куцко
Лидии Смирновой

ФУНКЦИОНАЛЬНОЕ ТЕСТИРОВАНИЕ

УЧЕБНОЕ ПОСОБИЕ

Содержание

ТЕМА 1.1 ВВЕДЕНИЕ.....	2
ТЕМА 1.2 ВИДЫ И МЕТОДЫ ТЕСТИРОВАНИЯ	7
ТЕМА 1.3 ПРИНЦИПЫ ДЕЛОВОЙ КОММУНИКАЦИИ.....	16
ТЕМА 1.4 ТЕСТИРОВАНИЕ ТРЕБОВАНИЙ И ДОКУМЕНТАЦИИ.....	19
ТЕМА 1.5 ПРИНЦИПЫ РАЗРАБОТКИ ТЕСТОВ	24
ТЕМА 1.6 УПРАВЛЕНИЕ ТЕСТАМИ, РЕГРЕССИОННОЕ ТЕСТИРОВАНИЕ	27
ТЕМА 1.7 СОЗДАНИЕ ОТЧЕТА О ДЕФЕКТЕ	30
ТЕМА 1.8 СОЗДАНИЕ ОТЧЕТА О РЕЗУЛЬТАТАХ ТЕСТИРОВАНИЯ	34
ТЕМА 2.1. ОСНОВЫ РАБОТЫ ВЕБ-ПРИЛОЖЕНИЙ.....	37
ТЕМА 2.2. ОСНОВЫ БАЗ ДАННЫХ И SQL	40
ТЕМА 2.3. ПЛАН ТЕСТИРОВАНИЯ. ОСОБЕННОСТИ РАБОТЫ В КОМАНДЕ	49
ТЕМА 2.4. ТЕСТИРОВАНИЕ ФОРМ, ТЕСТИРОВАНИЕ СОВМЕСТИМОСТИ	53
ТЕМА 2.5. ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА ТЕСТИРОВАНИЯ, ВВЕДЕНИЕ В АВТОМАТИЗИРОВАННОЕ ТЕСТИРОВАНИЕ	60
ТЕМА 2.6. ДОМЕННОЕ ТЕСТИРОВАНИЕ И ДРУГИЕ МЕТОДЫ ПРОЕКТИРОВАНИЯ ТЕСТОВ	64
ТЕМА 2.7. ОСНОВЫ ТЕСТИРОВАНИЯ ВЕБ-СЕРВИСОВ REST, ФОРМАТ ОБМЕНА ДАННЫМИ JSON.....	69
ТЕМА 2.8. ОСОБЕННОСТИ ТЕСТИРОВАНИЯ ВЕБ-СЕРВИСОВ, ФОРМАТ ОБМЕНА ДАННЫМИ XML.....	75
ТЕМА 3.1 ОЦЕНКА ТРУДОЗАТРАТ ПРИ ТЕСТИРОВАНИИ	79
ТЕМА 3.2 НЕПРЕРЫВНАЯ ИНТЕГРАЦИЯ, ОСНОВЫ КОМАНДНОЙ СТРОКИ	82
ТЕМА 3.3 ПОДГОТОВКА РАБОЧЕГО МЕСТА WINDOWS.....	87
ТЕМА 3.4 МЕТОДОЛОГИИ РАЗРАБОТКИ ПО	92
ТЕМА 3.5 ОСОБЕННОСТИ МОБИЛЬНОГО ТЕСТИРОВАНИЯ.....	95
ТЕМА 3.6 ОСОБЕННОСТИ ТЕСТИРОВАНИЯ ЛОКАЛИЗАЦИИ, ОСНОВЫ ТЕСТИРОВАНИЯ ДОСТУПНОСТИ	99

IT-Academy

Процесс/методология разработки ПО (software development process) – это набор задач и/или деятельности, которые решаются и/или происходят в ходе процесса создания программы.

Кардинально отличаются друг от друга следующие методологии разработки ПО:

Waterfall (каскадная, водопадная модель)

Ошибка, найденная на более поздних этапах жизненного цикла проекта, может привести к большим дополнительным расходам, даже срыву проекта.

Анализ требований

Спецификация

Проектирование ПО

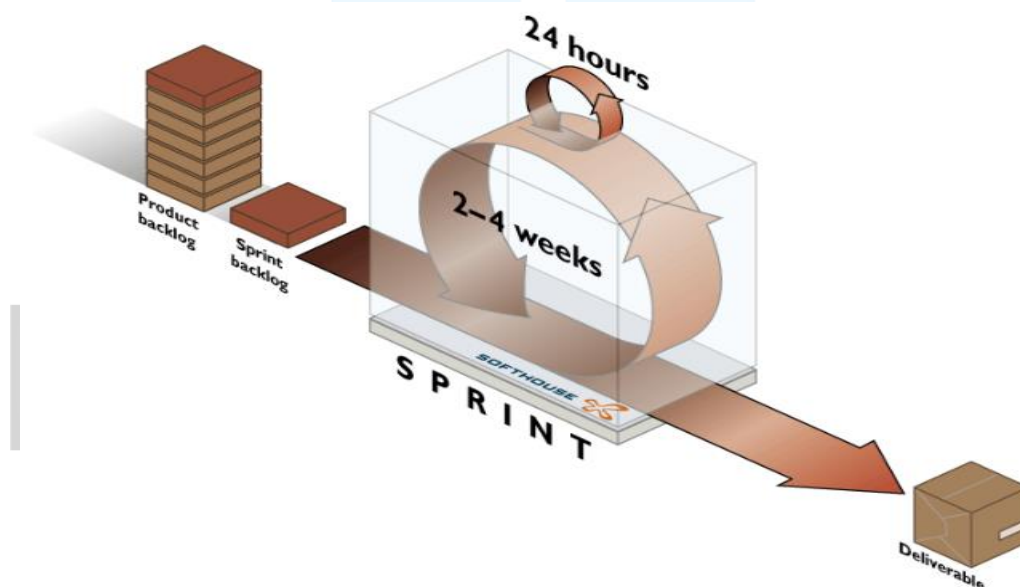
Программирование

Тестирование

Внедрение

Сопровождение

Agile (гибкая модель разработки)



Информационные технологии используются не только в производственных сферах, но и в быту. Поэтому тестирование ПО необходимо по следующим причинам:

- Пользователи склонны отдавать предпочтение качественным продуктам.
- Так как информационные технологии широко применяются в банковских и других сферах, пользователи не хотят рисковать личными данными, деньгами.
- Человек не желает рисковать здоровьем нации, планеты, жизнями людей и т.д.
- Увеличивается количество устройств и конкурентоспособных приложений.
- Возрастает сложность ПО.

Семь принципов тестирования:

- **Принцип 1. Тестирование демонстрирует наличие дефектов.** Тестирование может показать, что дефекты в программном обеспечении есть, но не может доказать, что никаких дефектов не существует. Но даже если никаких дефектов не найдено, это не доказывает правильность работы программы.
- **Принцип 2. Исчерпывающее тестирование невозможно.** Протестировать абсолютно всё (все комбинации входов и предусловий) не представляется возможным, за исключением тривиальных случаев. Вместо этого используются риски и приоритеты для эффективного поиска дефектов.
- **Принцип 3. Раннее тестирование.** Тестовые активности должны начинаться как можно раньше в цикле разработки ПО, чтобы быть эффективнее.
- **Принцип 4. Скопление дефектов.** Небольшое количество модулей содержит большинство дефектов, выявленных в ходе тестирования, или демонстрирует наибольшее количество операционных сбоев.
- **Принцип 5. Парадокс пестицида.** Если одни и те же тесты повторяются снова и снова, в конце концов с их помощью вы перестанете находить дефекты.
- **Принцип 6. Тестирование зависит от контекста.** Тестирование проводится по-разному в различных ситуациях.
- **Принцип 7. Заблуждение об отсутствии ошибок.** Нахождение и исправление дефектов не поможет, если разработанная система не удовлетворяет нуждам и ожиданиям пользователей.

Тестирование программного обеспечения (Software Testing) – процесс анализа программного средства и сопутствующей документации с целью повышения качества продукта.

Обеспечение качества (Quality Assurance) – мероприятия, охватывающие все этапы разработки, выпуска и эксплуатации ПО, проводимых на разных стадиях жизненного цикла ПО, для обеспечения качества выпускаемого продукта.

Контроль качества (Quality Control) – рабочие методы и активности, нацеленные на выполнение требований к качеству производимого продукта.

Верификация (Verification) – это процесс оценки системы или её компонентов с целью определения удовлетворяют ли требованиям, определенным в начале.

Валидация (Validation) – это определение соответствия разрабатываемого ПО ожиданиям и потребностям пользователя, требованиям к системе.

Тест-план (Test Plan) – часть проектной документации, описывающая и регламентирующая процесс тестирования.

Чек-лист (Check-List) – набор идей тестов, которые проверяют работу программы.

Тест-кейс (Test Case) – набор входных данных, условий выполнения и ожидаемых результатов, разработанный с целью проверки того или иного свойства или поведения программного средства.

Тестовый сценарий (Test Scenario, Test-Suite) – набор тест-кейсов, собранных в группу (последовательность) для достижения некоторой цели.

Отчет о тестировании (Test Result Report, TRR) – документ, подводящий итог проделанной работы в ходе тестирования, а также содержащий оценку состояния качества программы.

Артефакт (Test Artefact, Project artefact) – любой вид материального побочного производного, создаваемого при разработке программного обеспечения (например, чек-листы, тест-кейсы, сценарии использования, требования, справочные документы).

Билд (Build) – промежуточная версия (сборка) программного средства (финальный билд называют релизом (**release**)).

Тестовое окружение (Test Environment) – окружение, включающее в себя аппаратное обеспечение, измерительную аппаратуру, имитаторы, программный инструментарий и прочие технологии, необходимые для проведения теста.

Отладка (Debugging) – процесс поиска, анализа и устранения причин отказов в программном обеспечении.

Дефект (Defect, Bug) – любое несоответствие фактического и ожидаемого результата (согласно требованиям или здравому смыслу).

Ожидаемый результат (Expected Result) – такое поведение программного средства, которое мы ожидаем в ответ на наши действия.

Качество (Quality) – показатель степени соответствия продукта его требованиям.

Метрика качества (Quality Metric) – числовое значение некоторого показателя качества. Может определяться расчетным способом или по некоторой формуле.

Качество ПО включает 8 характеристик (ISO/IEC 25010:2011):

- Функциональную пригодность (Functional suitability).
 - Производительность (Performance efficiency).
 - Совместимость (Compatibility).
 - Удобство использования (Usability).
 - Надежность (Reliability).
 - Безопасность (Security).
 - Ремонтопригодность (Maintainability).
 - Переносимость (Portability).
-
-

Вопросы для обсуждения:

1. Почему Вы решили обучиться профессии тестировщика?
 2. Какие особенности тестирования ПО вы знаете?
 3. Какие задачи выполняет тестировщик на проекте?
 4. Какие на ваш взгляд самые сложные задачи у тестировщика?
 5. Опишите, что можно тестировать в приложениях.
-
-
-

ТЕМА 1.2 ВИДЫ И МЕТОДЫ ТЕСТИРОВАНИЯ

Уровни тестирования (по степени детализации компонентов системы):

- **Компонентное (модульное) тестирование** (Component Testing, Unit Testing) – тестирование отдельного модуля.
 - **Интеграционное тестирование** (Integration Testing) – проверка взаимодействия модулей.
 - **Системное тестирование** (System Testing) – проверка всего приложения.
-
-

Компонентами для тестирования могут выступать:

- Отдельный метод и/или функция в коде программы.
- Отдельная часть программы (регистрация, логин, аккаунт пользователя).
- Отдельная программа из сложной интеграционной системы.

Интеграционное тестирование (Integration testing) – это тестирование части системы или системы в целом, состоящей из двух и более частей, при этом основной задачей является поиск дефектов, связанных с ошибками взаимодействия между этими частями. По сути, проведение тестов при интеграционном тестировании – это проверка правильной коммуникации между двумя интерфейсами.

Интерфейс (Interface) – это совокупность способов и методов взаимодействия двух информационных систем, устройств или программ. Чаще всего встречаются такие интерфейсы как:

Интерфейс программирования приложений (API) – набор методов, которые можно использовать для доступа к функциональности другой программы.

Интерфейс командной строки (CLI) – инструкции компьютеру даются путём ввода с клавиатуры текстовых строк (команд).

Графический интерфейс пользователя (GUI) – программные функции представляются графическими элементами экрана.

Иногда, при необходимости тестировать приложение при отсутствии или недоступности некоторых частей системы, разработчикам приходится создавать для них **заглушки** (stub) и **имитаторы** (mock). На момент тестирования может не хватать как компонентов самой программы (например, тестирование процесса регистрации без существующей БД), так и подключаемых внешних систем или модулей (например, регистрация из социальных сетей без непосредственного подключения к ним).

Интеграционное тестирование можно подразделить на 2 подвида:

Компонентное интеграционное тестирование (component integration testing) – проверка взаимодействия между несколькими компонентами одного приложения.

Системное интеграционное тестирование (system integration testing) – это тестирование взаимодействия между всеми компонентами системы, между различными приложениями, объединенными в систему, интерфейсами связи с внешними системами (интернет и т.д.).

Приёмочное тестирование (Acceptance testing) – тестирование по отношению к потребностям и требованиям пользователя, проводимое с целью дать возможность пользователям, заказчикам определить, принимать систему или нет. По сути является подвидом системного тестирования, выполняется на заключительном этапе перед передачей продукта заказчику и/или конечным пользователям. Выделяют следующие типичные формы приемочного тестирования:

- Пользовательское приемочное тестирование (**UAT**).
 - Эксплуатационное приемочное тестирование (**OAT**).
 - Альфа- и бета- тестирование.
-
-

Пользовательское приемочное тестирование (User acceptance testing) – это тестирование конечного продукта заказчиком и/или конечными пользователями, которое может проходить как на оборудовании производителя, так и пользователей.

Эксплуатационное приемочное тестирование (Operational acceptance testing) – это тестирование заказчиком эксплуатационных характеристик системы, таких как резервное копирование/восстановление, аварийное восстановление и другие задачи технической поддержки, проверки уязвимостей безопасности.

Альфа-тестирование (Alpha testing) – тестирование потенциальными пользователями/заказчиками или независимой командой тестирования внутри организации разработчиков.

Бета-тестирование (Beta testing) – тестирование потенциальными и/или существующими клиентами/заказчиками на внешней стороне никак не связанными с разработчиками. Это форма внешнего приёмочного тестирования готового программного обеспечения для того чтобы получить отзывы рынка

Виды/направления тестирования (по целям и задачам):

- **Функциональное тестирование** (Functional testing) – тестирование, основанное на анализе спецификации функциональности приложения, проводимое с целью проверки на соответствие требованиям. Проверка того, «что» система делает.

- **Нефункциональное тестирование** (Non-functional testing) – тестирование свойств приложения, которые не относятся к функциональности системы. Проверка того, «как» работает приложение (надежность, эффективность, практичность, сопровождаемость).
-
-

Нефункциональные виды тестирования:

- **Инсталляционное тестирование** (Installation Testing) – тестирование, направленное на выявление дефектов, влияющих на установку/ обновление/ повторную установку/ удаление приложения.
-
-

- **Конфигурационное тестирование** (Configuration Testing) – тестирование, направленное на проверку работы программного обеспечения при различных конфигурациях системы (платформах, поддерживаемых драйверах, при различных конфигурациях компьютеров и т.д.).
-
-

- **Тестирование совместимости** (Compatibility Testing) – тестирование, направленное на проверку способности приложения работать в указанном окружении (заявленных браузерах, операционных системах, мобильных устройствах).
-
-

- **Тестирование графического интерфейса** (GUI Testing) – анализ соответствия графического пользовательского интерфейса программы спецификациям.
-
-

- **Тестирование удобства использования** (UX, Usability Testing) – тестирование, направленное на исследование того, насколько конечному пользователю понятно, как работать с продуктом, а также на то, насколько ему нравится использовать продукт.
-
-

- **Тестирование интернационализации** (Internationalization Testing, i18n) – тестирование, направленное на проверку готовности продукта к работе с использованием различных языков и с учётом различных национальных и культурных особенностей.
-
-

- **Тестирование локализации** (Localization Testing, l10n) – тестирование, направленное на проверку корректности и качества адаптации продукта к использованию на том или ином языке с учётом национальных и культурных особенностей.
-
-

- **Тестирование безопасности** (Security Testing) – тестирование с целью оценить защищённость программного продукта.
-
-

- **Тестирование доступности** (Accessibility Testing) – тестирование, направленное на исследование пригодности продукта к использованию людьми с ограниченными возможностями.
-
-

- **Тестирование производительности** (Performance Testing) – тестирование, проводимое с целью оценить поведение системы под нагрузкой.
-
-

- **Нагрузочное тестирование** (Load Testing) – тестирование, производимое при нагрузке в допустимых пределах и некотором превышении этих пределов.
-
-

- **Стресс тестирование** (Stress Testing) – тестирование, производимое при нагрузках, значительно превышающих расчётный уровень, или в ситуациях недоступности значительной части необходимых приложению ресурсов.
-
-

Виды/направления тестирования (по тестированию изменений):

- **Тестирование нового функционала** (New Feature Testing) – проверка того, что новый функционал работает верно.
 - **Повторное тестирование** (Retest, bug-fix verification) – проверка того, что поломанное действительно исправили.
 - **Регрессионное тестирование** (Regression Testing) – проверка того, что ранее работающий функционал по-прежнему работает после внесения изменений в код системы (исправление ошибки, реализация новой функции), а также после изменений в окружении. Во время тестирования проводится постоянно, например, во время проверки исправления ошибки нужно обращать внимание на работающую рядом функциональность.
-
-

Регрессионное тестирование является очень важной техникой тестирования и может проводиться на всех уровнях тестирования и применимо как к функциональному, так и нефункциональным видам тестированию. Регрессионные тесты должны запускаться и при изменениях в ПО, а также при изменениях в окружении.

Выбор тестов для регрессионного тестирования очень важен и требует определенной степени знания архитектуры приложения, его предполагаемого развития, а иногда и профессиональных особенностей разработчиков. Выбор зависит от множества условий и может основываться на:

- Риске обнаружения дефектов в ранее работавшем ПО.
 - Важности отдельных функций или модулей для бизнеса или безопасности.
 - Областях приложения, которые регулярно меняются.
 - Функциях или частях приложения с высокой вероятностью ошибок.
-
-
-

Говоря о **тестировании изменений**, верно утверждать, что:

- Если разработчики обещали сделать новую функциональность – не факт, что сделали и сделали правильно, нужно обязательно перепроверить.
 - Если разработчики обещали исправить ошибку – не факт, что исправили, нужно обязательно перепроверить.
 - Если разработчики говорят, что в этом билде нет серьезных изменений и ничего не случится – совершенно не факт, обязательно нужно проверить.
 - При исправлении старых ошибок могут появляться новые.
 - Если появилось что-либо новое, то программа должна быть протестирована повторно.
 - Если исправлена ошибка, то программа должна быть протестирована повторно.
 - Если ничего не менялось, но изменились внешние условия, то программа должна быть протестирована повторно.
 - Тесты должны разрабатываться с целью их повторного использования.
-
-
-

Виды/направления тестирования (в зависимости от фазы разработки).

- **Тестирование разработки** (Development testing) – тестирование, проводимое во время разработки системы, обычно в рабочей среде разработчиков. Основная цель – вызвать максимально возможное количество отказов, чтобы они были обнаружены и исправлены.
 - **Приёмочное тестирование** (Acceptance testing) – тестирование по отношению к потребностям пользователя или заказчика. Основная цель – подтвердить, что система работает, как ожидается, чтобы решить принимать или нет.
 - **Тестирование в период сопровождения** (Maintenance testing) – тестирование изменений в действующей системе или влияния изменений в окружении на действующую систему. Основная цель – проверить, что при разработке изменений не было произведено новых дефектов.
-
-
-

Главные **мотивы тестирования в период сопровождения**:

- Плановое изменение программных продуктов с целью улучшения, изменения окружения (обновление БД, патчи для уязвимостей ОС).
 - Аварийные изменения, исправления ошибок, найденных пользователями.
 - Миграция программных продуктов (из одной среды в другую).
 - Эксплуатационное тестирование в новых условиях.
 - Вывод из эксплуатации ПО или частей системы.
 - Миграция данных или архивирования (для длительного хранения).
-
-

Можно выделить следующие **особенности тестирования** в этот период:

- Изменения в приложении как правило незначительные, поэтому в рамках тестирования изменений преобладает регрессионное тестирование, объем которого будет основываться на рисках поломок, размером существующей системы и масштабах изменений.
 - Тестируемое приложение может быть в использовании 24/7 и быть жизненно важным для бизнеса.
 - Система должна быть протестирована быстро и эффективно, чтобы с одной стороны не пропустить ошибки, а с другой не быть препятствием для успешного ведения бизнеса.
 - Очень часто на таких проектах или полностью отсутствует требования к приложению, или они старые и уже не актуальны. Тогда тестировщику приходится пользоваться уже имеющейся системой (в рамках эталонного тестирования), руководством пользователя или знаниям коллег.
-
-

Уровни функционального тестирования (по степени важности тестируемых функций и приоритету выполнения тестов):

- **Дымовое тестирование** (Smoke Testing) – тестирование «только самого важного», без успешного завершения которого не имеет смысла переходить на следующий уровень тестирования.
 - **Тестирование критического пути** (Critical-Path Testing) – тестирование часто используемых повседневных функций приложения, то есть функций, которые выполняет бизнес-пользователь чаще всего в реальной жизни.
 - **Расширенное тестирование** (Extended Testing) – тестирование всех возможных функций приложения.
-
-

Внимание! Возможна путаница!

Единой классификации не существует, и две категории имеют в обиходе профессионалов похожие названия:

- Уровни тестирования (по степени детализации приложения) = компонентное, интеграционное, системное.
- Уровни функционального тестирования (по важности функций) = smoke, critical path, extended.

Виды/направления тестирования (по принципу работы с приложением):

- Позитивное тестирование (Positive Testing).
 - Негативное тестирование (Negative Testing).
-
-

Виды/направления тестирования (по запуску кода на исполнение):

- **Статическое** (Statistical Testing) – тестирование без запуска программы. Например, тестирование документации, статический анализ кода приложения специальными инструментальными средствами.
 - **Динамическое** (Dynamic Testing) – тестирование с запуском программы.
-
-

Методы тестирования (по знанию и использованию кода тестирующим):

- **«Белого ящика»** (White-Box Testing, Glass-Box Testing) – тестирующий имеет доступ к коду, опирается на понимание кода во время составления тестов и/или использует код во время тестирования.
 - **«Черного ящика»** (Black-Box Testing) – тестирование приложения без знания внутренней структуры системы (кода), когда тестирующий опирается на требования и работает с интерфейсом приложения как конечный пользователь.
 - **«Серого ящика»** (Grey-Box Testing) – тестирование, при котором тестирующий имеет доступ только к некоторой части кода (например, доступ к БД) и использует это при составлении тестов и их выполнении.
-
-

Виды/направления тестирования (по степени формализации тестов):

- **На основе тест-кейсов** (Scripted testing, test case based testing) – тестирование производится на основе заранее подготовленных тест-кейсов.
 - **На основе чек-листов** (Checklist-based testing) – тестирование производится на основе чек-листов.
 - **Исследовательское тестирование** (Exploratory testing) – неформальный метод проектирования тестов, при котором тестирующий активно продолжает проектирование тестов в то время, как эти тесты выполняются, и использует полученную во время тестирования информацию для проектирования новых и улучшенных тестов.
 - **Свободное тестирование** (Ad-hoc testing) – тестирование, выполняемое без формальной подготовки тестов, формальных методов проектирования тестов, определения ожидаемых результатов и руководства по выполнению тестирования.
-
-

- **Хаотическое тестирование** (Monkey testing) – тестирование случайным выбором из большого диапазона входов, случайным нажатием кнопок, без соотнесения с тем, как в реальности будет использоваться система.
-
-

Рекомендации и замечания:

- Необходимо очень детально анализировать все уровни требований системы (бизнес-требования, пользовательские и др.), чтобы на начальном этапе тестирования разработать правильную стратегию тестирования, то есть выбрать подходы, типы, методы и направления тестирования, которые помогут обеспечить качество продукта.
 - В различных информационных источниках типы, виды, подходы, методы могут иметь иные названия, то есть уровни могут называться типами, методы – направлениями. Важно не теряться, а понимать, на основе чего проводится классификация.
 - При тестировании приложения типы, виды, подходы, методы, направления могут дополнять друг друга, например, инсталляционное тестирование проводится в рамках тестирования совместимости, также для него можно выполнить дымовое тестирование, критического пути и расширенное, часто параллельно с регрессионным тестированием.
 - Если Вы понимаете, что в стратегию тестирования не внесли определенный вид тестирования, который необходимо проводить для выпуска качественного продукта, стоит как можно быстрее поднять этот вопрос для обсуждения в команде по тестированию, согласовать с проектным менеджером, заказчиком (зависит от процесса на проекте), чтобы выполнить тестирование в полной мере.
-
-
-

Вопросы для обсуждения и задания:

1. Приведите примеры проверок, которые можно отнести к smoke, critical-path, extended тестированию для любого приложения на ваш выбор.
 2. Если Вы планируете тестировать приложение по бухгалтерии для компании, в которой работают 20 сотрудников и расширение штата не планируется, то какие подходы, виды, типы, уровни и т.д. выберите для тестирования?
 3. Если Вы планируете тестирование социальной сети для людей пожилого возраста из разных стран, то какие виды, типы, уровни, подходы и т.д. будете использовать во время тестирования, чтобы обеспечить качество?
 4. Как узнать, на каких устройствах или операционных системах, браузерах нужно проводить тестирование приложения?
-
-
-
-

Уровни тестирования:	<ul style="list-style-type: none"> • Компонентное тестирование • Интеграционное тестирование • Системное тестирование • Приемочное тестирование • Альфа- и бета- тестирование
Виды/направления функциональные виды тестирования	<ul style="list-style-type: none"> • Инсталляционное тестирование • Конфигурационное тестирование • Тестирование совместимости • Тестирование графического интерфейса • Тестирование удобства использования • Тестирование интернационализации • Тестирование локализации • Тестирование безопасности • Тестирование адаптации для людей с ограниченными возможностями • Нагрузочное и стресс тестирование
По тестированию изменений	<ul style="list-style-type: none"> • Тестирование нового функционала • Повторное тестирование • Регрессионное тестирование
В зависимости от фазы разработки	<ul style="list-style-type: none"> • Тестирование разработки • Приемочное тестирование • Тестирование в период сопровождения
по степени важности тестируемых функций и приоритету выполнения тестов	<ul style="list-style-type: none"> • Дымовое тестирование • Тестирование критического пути • Расширенное тестирование
По принципу работы с приложением	<ul style="list-style-type: none"> • Позитивное тестирование • Негативное тестирование
по запуску кода на исполнение	<ul style="list-style-type: none"> • Статическое • Динамическое
Методы тестирования	<ul style="list-style-type: none"> • «Белого ящика» • «Черного ящика» • «Серого ящика»
По степени формализации тестов	<ul style="list-style-type: none"> • На основе тест-кейсов, на основе чек-листов • Исследовательское тестирование • Свободное (интуитивное) тестирование

ТЕМА 1.3 ПРИНЦИПЫ ДЕЛОВОЙ КОММУНИКАЦИИ

Основные способы коммуникации в команде:

- Личные беседы (один на один).
- Совещания (собрания, “митинги”).
- Почта.
- Мгновенные сообщения.
- Звонки.

Правила работы с почтой:

- Письма должны быть отправлены вовремя.
- Письма должны иметь корректный список получателей (То – лица, которым адресовано письмо, Сс – лица, от которых не ожидается ответ на письмо, но которые должны быть проинформированы о происходящем в переписке).
- Письмо должно содержать осмысленный информативный заголовок, который кратко, но емко отражает суть письма.
- Ответы на письмо или письма на одну и ту же тему стоит отправлять в одном потоке, то есть в одной цепочке писем, используя функцию «Ответить», не создавая отдельное письмо.
- Наполнение письма должно быть корректное, лаконичное, структурированное, что позволяет легко и быстро понять проблему, вопрос (если таблицы или графики визуализируют информацию, то их можно использовать в письме).
- Корректные грамматика, пунктуация, синтаксис и т.п.
- Аккуратное оформление (включая цвета, шрифт). Это означает, что очень важные детали письма можно выделить другим цветом или жирным шрифтом, но не стоит использовать много разных цветов и шрифтов в одном письме.
- Используем общепринятую структура письма (приветствие, тело, завершение, подпись).
- Помним, что нужно прикрепить к письму необходимые документы, особенно если на них идет ссылка в содержании письма.
- Не прикрепляем к письму множество документов или тяжеловесные документы. В таком случае лучше использовать ссылки на облачные хранилища.
- Верно установленный показатель важности (помним, что добавляем высокую важность только для действительно важных писем).
- Отвечаем на письма в течение суток после их получения. Если письмо с высокой срочностью, то отвечаем в течение ожидаемого отправителем времени. Если для ответа на письмо необходимо выполнить действия, что занимает продолжительное время, то стоит ответить, что задание получено, что Вы приступаете к выполнению с указанием запланированного времени, к которому Вы отправите ответ.
- Необходимо каждое письмо прочитать, категоризировать, удалить или заархивировать (создать папки для хранения писем по категориям).
- Помним про автоматический ответ на полученные письма во время отпуска, болезни с указанием ответственного лица, к которому можно обратиться вместо Вас по срочным вопросам.
- Отвечать нужно на письмо, которое адресовано Вам лично:
 - если Вы состоите в большом списке получателей и Вам не ясна Ваша роль (стоит уточнить у отправителя, какие действия от Вас требуются);

- если Вы стоите в То или в Сс и Вам есть что добавить.
 - Используйте адекватные имена для профиля и почтового адреса (например, pupsik93@mail.ru не поможет приобрести авторитет среди коллег).
-
-
-
-

Часто используемые аббревиатуры в работе с почтой:

MFU – Meeting Follow Up.

FYI – For Your Information.

IFU – Interview Follow Up.

SIN – Service Interruption Notification.

OOO – Out Of Office.

EOD – End Of the Day.

Правила работы с мгновенными сообщениями (Skype, Skype for Business, Viber, Slack, Microsoft Teams, Discord, etc):

- Используйте инструменты мгновенных сообщений по вопросам, которые необходимо решить срочно или ответ занимает короткое время.
 - Если важный вопрос отправлен на почту, то можно написать адресату в Skype с просьбой посмотреть письмо, которое Вы отправили на почту.
 - Если инструмент мгновенных сообщений имеет онлайн индикатор присутствия, то выставляйте его в актуальное положение. Например, если Вас нет на месте, то замените индикатор online в Skype на away, чтобы собеседник понимал, что не стоит ждать от вас мгновенной реакции.
 - Используйте адекватное имя профиля в инструментах мгновенных сообщений.
 - Не пишите просто «Привет» в чате.
-
-
-
-

Правила организации и проведения совещания (собраний, «митинга»):

- Подготовьте план совещания.
- Заранее разошлите план совещания со списком вопросов, которые будут обсуждаться на совещании (собрании, «митинге»), всем участникам.
- Выберите наиболее удобное для всех участников время и место.
- Зарезервируйте комнату.
- Придите на место проведения заранее, подготовьте комнату и оборудование.
- Если собрание отменяется, оповестите всех участников по возможности заранее и внесите изменения в календарь загруженности комнаты.
- Соглашения, достигнутые в устной беседе, необходимо задокументировать и разослать MFU.
- Во время собрания делайте пометки, не рассчитывайте на свою память, так как практика показывает, что без пометок важных моментов можно что-то забыть после собрания (meeting notes, minutes).
- Фиксируйте адресованные вам задачи во время митинга (action items).

- Проводите оценку на основе объективных критериев. Руководствуйтесь фактами. Выслушайте аргументы собеседника. Не скрывайте информацию, которая может быть полезна в работе вашим коллегам.
- Проясняйте непонятное, задавайте вопросы, не перебивая собеседника. Не рассчитывайте на то, что “непонятное” на собрании станет “ясным” после него.
- Собрание стоит проводить только тогда, когда есть вопросы для обсуждения. Рационально используйте свое время и время коллег.
- Избегайте бессмысленных разговоров.

Правила организации и проведения звонков (удаленных собраний):

- Проверяйте связь перед звонком, работоспособность и настройки устройств (микрофон, наушники).
- Подготовьтесь к демонстрации экрана. Например, если во время звонка заказчику планируете задавать вопрос с демонстрированием приложения, то заранее подготовьте нужные страницы приложения и данные, чтобы не искать ссылки, не логиниться, не выполнять дополнительные подготовительные шаги, которые будут занимать время заказчика.
- Выключайте микрофон, если не говорите, чтобы не создавать дополнительный шум.

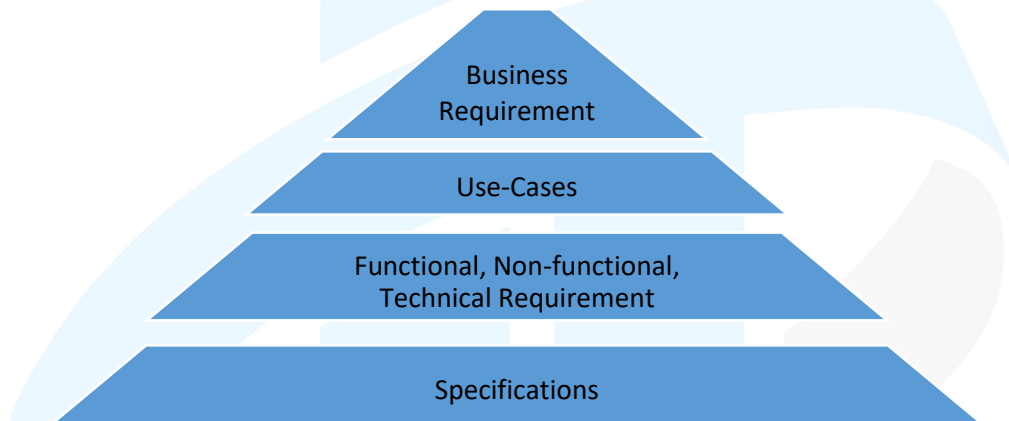
Вопросы для обсуждения и задания:

1. Что делать, если разработчик не хочет исправлять ошибки?
2. Если заказчик отвечает письмами на вопросы по требованиям, но ответы не помогают прояснить их суть. Какие действия можно предпринять в такой ситуации?
3. Ведущий тестировщик Иванов в кабинете при всех коллегах на повышенных тонах обсуждает плохую производительность тестировщика Сидорова. Какие ошибки в коммуникации допущены и что стоит предпринять участникам конфликта в данной ситуации?
4. Заказчик за чашкой кофе в холле офиса ответил на вопрос тестировщика по требованиям приложения. Какими будут следующие действия тестировщика?

ТЕМА 1.4 ТЕСТИРОВАНИЕ ТРЕБОВАНИЙ И ДОКУМЕНТАЦИИ

Требования (Requirements) – это документ, описывающий то, что должно быть реализовано в приложении или программе. В нем описано поведение системы, свойства системы и/или ее атрибуты.

Выделяют следующие уровни требований:



- **Уровень бизнес-требований** (Business Requirement) – «общее видение системы», то есть основная концепция приложения (Vision). Например, нужно разработать приложение, которое увеличит продажи благодаря возможности просматривать каталог продукции и заказывать товары онлайн.

-
-
-
- **Уровень пользовательских требований** (User Requirements, Use-Cases) – «что можно будет делать», то есть варианты использования системы. Например, в приложении можно создать личный кабинет, в котором будут размещаться истории заказов, можно посмотреть каталог всей продукции по категориям, отмечать понравившиеся товары, добавлять в корзину, делать заказ, оплачивать его онлайн.

-
-
-
- **Уровень функциональных и нефункциональных требований** (Functional, Non-functional, Technical Requirement) – «что конкретно должна выполнять система и каким образом она должна это выполнять». Например, какие страницы в приложении, какие вкладки, как выглядят вкладки, какие переходы между вкладками (в нашем примере – на странице оформления заказа после заполнения всех необходимых полей становится доступной кнопка оплаты, которая переводит на форму для ввода карты и т.д.).
-
-

- **Детализация требований в спецификации (Specifications)** – «система должна соответствовать таким-то параметрам», то есть требованиям, уточненным до числовых характеристик. Например, какие поля должны быть на странице заказа, в какой последовательности они должны располагаться на странице, какие допустимые значения для ввода в поля, какие поля являются обязательными, каковы их точные размеры.
-
-

Вопросы, ответы на которые необходимо знать в начале проекта при работе с требованиями:

- Какие источники требований у нас есть?
 - Где хранятся требования?
 - Кто ответственный за работу с требованиями на проекте?
 - Кто помещает требования в официальное хранилище?
 - Что является более корректным – изначальные спецификации, последующие письма, прототип?
 - Как мы узнаем об изменениях в требованиях?
 - Кто утверждает окончательные требования?
 - Кому мы можем задавать вопросы по требованиям?
 - Если нам не отвечают, кого спрашивать следующим, кого в конечном итоге?
 - Как мы можем предложить внести изменения в требования?
 - Кому и как мы должны сообщить о проблемах с требованиями?
-
-

Пути выявления требований:

- Интервью.
-
-

- Совещания.
-
-

- Наблюдение.
-
-

- Самостоятельное описание.
-
-

- Прототипирование.
-
-

- Анкетирование.
-
-

Если заказчик на словах пояснил требования, не конкретизировал их документально в понятном виде для команды, то имеет смысл самостоятельно законспектировать требования, то есть создать документ, который отразит детально то, что нужно сделать и как это будет работать, выглядеть, чтобы перестраховать себя и команду от ситуации, когда заказчик сказал «одно», а имел в виду «другое». Также стоит учитывать конкретную ситуацию, так как не всегда есть время и ресурсы на создание документации.

Свойства хорошего требования:

- **Завершенность** (Completeness).
- **Непротиворечивость** (Consistency).
- **Корректность** (Correctness).
- **Недвусмысленность** (Unambiguousness).
- **Проверяемость** (Verifiability).
- **Модифицируемость** (Modifiability).
- **Прослеживаемость** (Traceability).
- **Проранжированность** (Ranked for importance, Stability, Priority).

В английском языке есть много **слов-индикаторов**, наличие которых в требовании должно насторожить тестировщика:

Adequate, be able to, easy, provide for, as a minimum, be capable of, effective, timely, as applicable, if possible, TBD, as appropriate, if practical, at a minimum, but not limited to,

capability of, capability to, normal, minimize, maximize, optimize, rapid, user-friendly, simple, often, usual, large, flexible, robust, state-of-the-art, improved, efficient.

Техники тестирования требований:

- **Взаимный просмотр (peer review).**
 - **Вопросы.**
 - **Тест кейсы.**
 - **Рисунки, схемы и т.д.**
-

Рекомендации и техники по работе с требованиями:

- Начинаем анализировать (тестировать) требования как можно раньше, так как ошибка, найденная в требованиях на ранней стадии проекта дешевле в исправлении, чем такая же ошибка, найденная на конечной стадии проекта. А также хорошо протестированные требования позволяют сократить количество ошибок, допущенных разработчиками.
 - Несколько раз вдумчиво читаем. Помечаем непонятные места. Не задаем вопросы, пока не дочитали до конца и второй раз, так как дальше в требованиях может быть ответ на вопрос.
 - Пытаемся самостоятельно найти ответы на вопросы в Google (например, некоторые незнакомые аббревиатуры), уточняем у других членов команды (возможно, они разобрались в этом требовании).
 - Рисуем диаграммы, схемы, интеллект-карты, которые помогут структурировать информацию, составить связи между сущностями приложения, найти пропущенные детали требований или ответы на возникшие вопросы.
 - Составляем чек-лист, тест-кейсы (во время составления тест-кейсов, как правило, появляются дополнительные вопросы, ответов на которые не хватает в требованиях).
 - Задаем четкие, последовательные, структурированные вопросы человеку, который компетентен в этом и ответственен за требования на проекте (бизнес-аналитику, заказчику и др.).
 - Следует уточнять даже такие мелочи, как обязательность полей для заполнения, чувствительность к регистру текста, допустимая длина, разрешенные/запрещенные символы, поддерживаемые языки, значения по умолчанию, точные тексты сообщений.
 - Расценивайте полученный ответ как новое требование, как можно быстрее актуализируйте свои тесты, согласно новым данным, если вы не получили нужное представление о интересующем вас вопросе, то задайте дополнительные вопросы (иногда нужно 5+ циклов вопрос/ответ, чтобы решить проблему).
-
-

Требования могут быть представлены в виде:

- Последовательного, структурированного описания системы.
- User Story.
- Диаграмм.
- Блок-схем.
- Mock-ups (отрисованных страниц приложения).
- Старой работающей системы.
- Style Guide (описания стандартов графического интерфейса).

Вопросы для обсуждения и задания:

1. Приведите примеры плохого требования и какому свойству хорошего требования оно не соответствует?
2. Что делать, если Вы не понимаете требования?
3. Зачем необходимо тестировать требования?
4. Какую документацию можно тестировать на проекте?

IT-Academy

ТЕМА 1.5 ПРИНЦИПЫ РАЗРАБОТКИ ТЕСТОВ

Документация для тестирования:

- **Чек-лист (Check-List)** – набор идей тестов.
 - **Тест-кейс (Test Case)** – набор входных данных, условий / шагов выполнения и ожидаемых результатов, разработанный с целью проверки свойства или поведения программного средства.
 - **Тестовый сценарий (Test Scenario, Test Suite)** – набор тестов (тест-кейсов), собранных в последовательность для достижения некоторой цели.
-
-

Последовательность разработки и выполнения тестов:

- Простые позитивные.
 - Простые негативные.
 - Сложные позитивные.
 - Сложные негативные.
-
-

План разработки тестов:

- Разбиваем приложение на небольшие функциональные части (модули) и начинаем создавать тесты как можно раньше.
 - Составляем чек-лист, который покрывает требования идеями проверок.
 - Пишем вопросы, уточняем детали.
 - Создаем тест-кейсы, основываясь на чек-листах.
 - Обновляем тесты, как только обнаружили ошибку или изменилась функциональность.
-
-

Атрибуты тест-кейса*:

- Идентификатор (ID).
- Приоритет (Priority).
- Ссылка на требование (Requirement).
- Модуль / подмодуль (Module / Submodule).
- Описание (Description):
 - Название тест-кейса (Title)
 - Предусловие (Precondition) – опциональный атрибут
 - Шаги воспроизведения (Test Steps/Test procedure)
 - Постусловие (Postcondition) – опциональный атрибут
- Ожидаемый результат (Expected Result).

*** Важно: атрибуты тест-кейса отдельной компании или проекта могут сильно отличаться!**

Пример Тест-кейса (только некоторые атрибуты тест-кейса приведены для примера):

ID	Priority	RQ	Module	Sub module	Description	Expected result
C12	Extended test (A/B/C)	R97	Галерея	Загрузка файла	<p>Галерея, загрузка файла, имя со спецсимволами</p> <p>Приготовления: создать непустой файл с именем #\$\$%^&.jpg и размером от 1 до 4 Mb.</p> <p>1. Запустить gallery.exe</p> <p>2. Кликнуть кнопку «Загрузить картинку».</p> <p>3. Выбрать из списка приготовленный файл.</p> <p>4. Нажать кнопку «ОК».</p> <p>5. Нажать кнопку «Добавить в галерею».</p>	<p>1. Открывается главное окно галереи.</p> <p>2. Появляется диалоговое окно выбора файла для загрузки.</p> <p>3. Имя выбранного файла появляется в поле «Файл».</p> <p>4. Диалоговое окно файла закрывается, в поле «Файл» появляется полное имя файла.</p> <p>5. Выбранный файл появляется в верху списка файлов галереи.</p>

Для чего составляются тест-кейсы:

- Структурированный подход к тестированию.
- Возможность найти пропуски в требованиях.
- Планирование регрессионного тестирования.
- Передача знаний о приложении новым членам команды по тестированию.
- Удовлетворение требования заказчика.
- Обеспечение хранения информации.
- Контроль прогресса тестирования.

Тест-кейсы можно создавать / хранить в таких автоматизированных инструментальных средствах, как Excel, Google Documents, Test Link, TestRail, QC HP, JIRA plugins и др.

Рекомендации и замечания:

- Начинаем с простых и очевидных тестов, проверяя самые используемые функции приложения. Далее переходим к более сложным.
- Помним про логику работы приложения, сначала проверяем функционал.
- Если остается время, проводим свободное (ad-hoc) тестирование.
- Тесты не должны быть избыточны по отношению к другим тестам.
- Тесты не должны выполнять ненужные действия, а должны быть направлены на поиск ошибки.
- Составляем понятные тесты, предполагая, что их будет использовать другой человек при тестировании.
- Хорошее правило, когда актуальная тестовая документация хранится в общем доступе для всех участников проекта.
- Необходимо использовать удобные средства создания, хранения и прохождения тест-кейсов.

Вопросы для обсуждения и задания:

5. Назовите атрибуты (поля) тест-кейса?
6. В чем отличие чек-листа от тест-кейса?
7. Можно ли тестировать приложение, не составляя документацию (чек-лист, тест-кейсы)?
8. Для чего стоит составлять тест-кейсы?

IT-Academy

ТЕМА 1.6 УПРАВЛЕНИЕ ТЕСТАМИ, РЕГРЕССИОННОЕ ТЕСТИРОВАНИЕ

Регрессионное тестирование (Regression Testing) – проверка того, что то, что работало, по-прежнему работает после внесения изменений в код системы (исправление ошибки, реализация новой функции).

Существуют три вещи, которые должны нас побуждать к регрессии:

- **Регрессия багов (bug regression)** – с одной стороны мы должны убедиться, что все дефекты у нас закрыты к моменту релиза, а с другой – у нас не должно быть ситуации, когда дефекты возвращаются к нам в своем прежнем облики в новом билде, хотя в старом их по-прежнему нет.
- **Регрессия старых исправлений (old fix regression)** – этот вариант встречается реже и как правило, речь идет о фиксах, которые были когда-то сделаны и есть риск того, что проблемы могут возникнуть снова.
- **Общая регрессия (general functional regression)** – проверяем, что старый функционал работает так, как он работал, при этом мы не концентрируемся на отдельных частях, а идем по всему функционалу.

Выделяют 2 способа проведения регрессионного тестирования:

- Подтверждающий (verification) – раз за разом выполняем одни и те же тесты, которые выбраны как критичные.
- Осматривающий (look around) – каждый раз набор тестов изменяется, основываясь на риске возникновения ошибок в той или иной области.

Два главных составляющих, которые определяют необходимость прохождения теста – это **вероятность возникновения** и **сила влияния** потенциального дефекта:

- Вероятность возникновения – выше в областях, связанных с изменяемыми функциями или модулями, наиболее технически сложных.
- Сила влияния – больше в часто используемых функциях, или наоборот в редких, но при этом очень критичных, без которых встанет работа.

В зависимости от контекста проекта стоит определиться в каком виде хранить тестовую информацию – в виде чек-листов или подробных тест-кейсов.

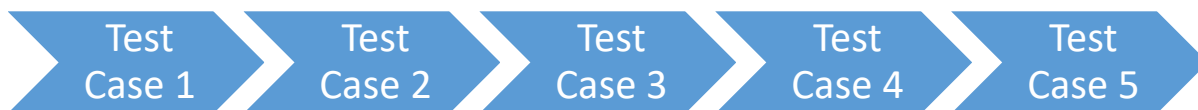
Создание чек-листов для тестирования	
Преимущества	Недостатки
Нивелирование «эффекта пестицида» в регрессионном тестировании.	Начинающие тестировщики не всегда эффективно проводят тесты без достаточно подробной документации.
Расширение тестового покрытия за счет отличий при прохождении.	Чек-листы невозможно использовать для обучения начинающих сотрудников.
Сокращение затрат на создание и поддержку тестов: не нужно писать много текста.	Заказчику или руководству может быть недостаточно того уровня детализации, который предлагают чек-листы.
Отсутствие рутины, которую так часто не любят квалифицированные тестировщики.	Необходимо дополнительное время на объяснение логики приложения новому тестировщику, так как чек-лист может быть непонятен на начальном этапе без пояснения приложения.

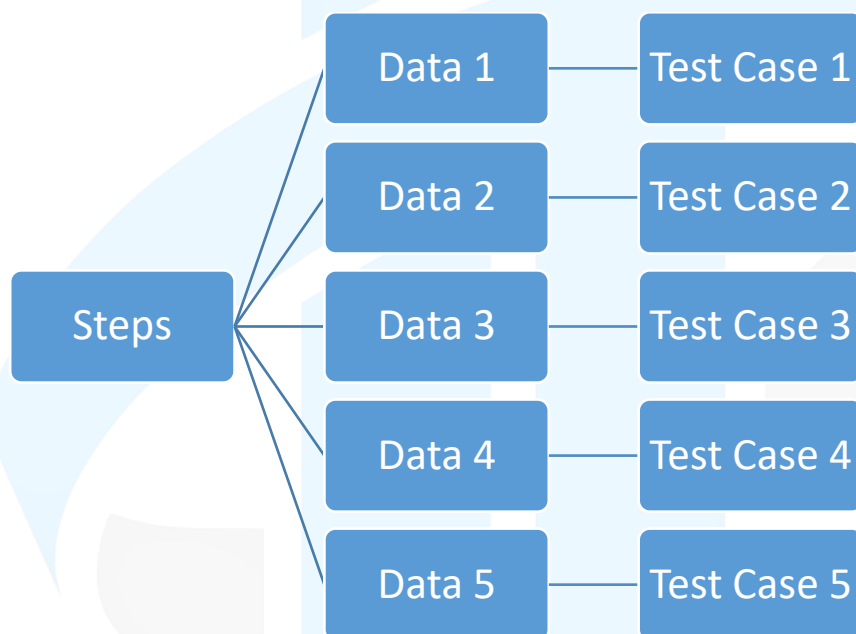
Выбор тестовой документации	
Чек-лист	Тест-кейс
Небольшой несложный проект	Сложный проект
Нехватка времени	Требуется повышенный контроль за тестированием и разнообразная отчетность
Стабильная квалифицированная команда	Много начинающих сотрудников или часто меняющаяся команда

Тестовый набор (test suite) – набор тестов (тест-кейсов), собранных в последовательность для достижения некоторой цели.

Test suite следует некоторой логике, например:

- Типичной последовательности использования приложения.
- Распределению функций по модулям.
- По тестированию функции с различными условиями или данными.





Вопросы для обсуждения и задания:

1. В каких случаях нужно проводить регрессионное тестирование?
2. Приведите примеры случаев, когда код приложения не менялся, но нужно провести регрессию?
3. По какому принципу надо выбирать тесты для регрессии?
4. У вас есть 2000 тестов и 1 день на проведение регрессии, какие тесты будете проводить?
5. В каких случаях нужно писать тест-кейсы, а в каких достаточно чек-листа?
6. По каким принципам можно организовывать тест-сьюты?

ТЕМА 1.7 СОЗДАНИЕ ОТЧЕТА О ДЕФЕКТЕ

Дефект (Defect) – любое несоответствие требованиям или функциональным спецификациям (здоровому смыслу). То есть это отклонение фактического результата (**Actual Result**) от ожидаемого результата (**Expected Result**), которое фиксируется в документе под названием “отчет о дефекте” (**Bug Report**), где содержится детальная информация об ошибке.

Bug report как правило создается в **баг-трекинговой системе** (JIRA, Rally, HP QC, Bugzilla, etc.)

Цель создания отчета об ошибке:

- Предоставить информацию о проблеме.
 - Приоритезировать проблему.
 - Учесть и проанализировать качество продукта.
 - Помочь программистам исправить.
-

Следующие сущности в баг-трекинговой системе могут быть как самостоятельными видами, так и синонимами баг-репорту:

- Improvement – запрос, на улучшение существующих функций.
 - Feature – запрос, на расширение ПО новой функцией или изменение существующей функциональности.
 - Change Request – запрос, для настройки или изменения системы.
 - Enhancement – запрос, на воплощение новых идей, нового поведения или новой функциональности ПО.
-

Написание отчета об ошибке состоит из описания трех частей:

- Что мы сделали (steps to reproduce the problem).
 - Что мы получили (actual result).
 - Что ожидали получить (expected result).
-

Атрибуты отчета о дефекте:

- Идентификатор (id).
-
- Краткое описание (summary).
-

- **Подробное описание (description).**

- **Ссылка на требование (requirement id).**

- **Фактический результат (actual result).**

- **Ожидаемый результат (expected result).**

- **Шаги воспроизведения (steps to reproduce).**

- **Важность (severity).**

- **Срочность (priority).**

- **Приложения (attachments) (e.g. screenshots, video recordings, log files, etc.).**

- **Воспроизводимость (reproducible).**

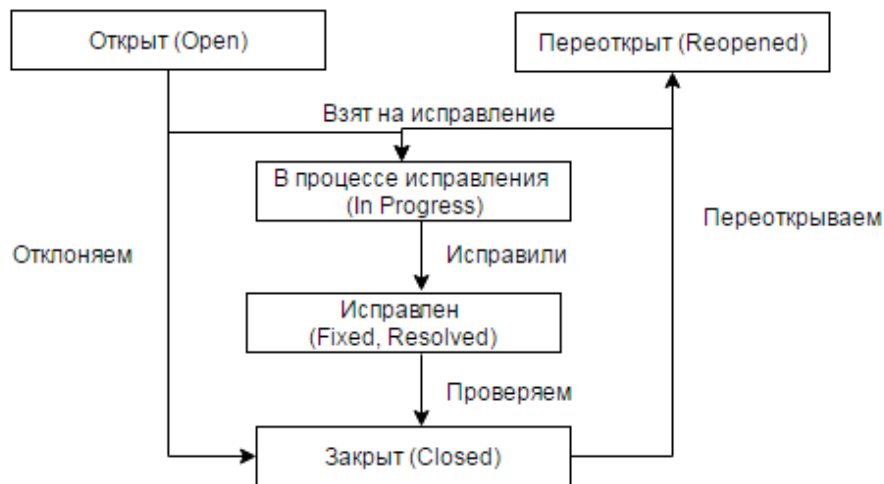
- **Симптом (symptom).**
- **Возможность «обойти баг» (workaround).**
- **Статус (Status)**

Могут встречаться разнообразные дополнительные атрибуты, которые необходимы в зависимости от процесса на проекте: ссылка на тест-кейс, юзе-стори, вопрос и т.д. (Requirement, Test-Case, Story, Question, etc., например, в JIRA можно создавать связи между багами и требованиями, то есть “линковать” баги на требования).

- **Дополнительная информация (additional information).**
- **Автор ошибки (reporter).**
- **Билд нахождения бага (Build).**
- **Билд исправления бага (Fixed Build).**
- **Кто ответственный за баг (Assignee).**
- **Отметки (Labels) и прочие.**

Жизненный цикл бага:

Самый простой пример жизненного цикла бага, который может меняться в зависимости от процесса разработки на проекте, выглядит следующим образом:



Могут быть добавлены статусы, например, “Заблокирован” (Blocked), “Отклонен” (Declined), “Назначен” (Assigned) и другие.

Пример бага из JIRA:

[Guidelines>Backend] Unclear warning message appears next to "Category" field when user is trying to save new guideline

Web browser: Google Chrome Version 56.0.2924.87

Description

Preconditions: Click on the [Gear] icon>choose [Backend].

Steps to reproduce:

1. Navigate to **Guidelines** (in header).
2. Choose **Guideline Categories**
3. Click **+New**
4. Fill in the [Name] field with no more than 20 characters.
5. Navigate to **Guidelines** (in header).
6. Choose **Guidelines List**
7. Click **+New**
8. Select the newly created category from drop-down list.
9. Click [Save]

Actual result: Warning message appears next to **Category** field. But user does not understand what it exactly means.

Expected result: Warning message should contain more clear text so that user could figure out what one is doing wrong.

Dates

Created:

Updated:

Collaborators

Agile

[View on Board](#)

Рекомендации и замечания:

- Критичный баг, найденный тестировщиком в последний день, является багом в работе самого тестировщика.
- Воспроизведите ошибку несколько раз, чтобы убедиться, что описываете правильные шаги и нет других зависимостей в появлении ошибки.
- Тщательно объясните, как воспроизвести ошибку, описывайте подробно, чтобы понятно было не только Вам. Помним, что исправленный баг придется проверять нам, возможно, через долгий промежуток времени. И отсутствие деталей помешает пройти шаги, получить ожидаемый результат и убедиться, что баг исправлен.
- Пишите отчёт понятно.
- Давайте ссылку на требование (если это возможно).
- Указывайте окружение.
- Скриншоты и видеозаписи экрана могут упростить и ускорить понимание шагов воспроизведения и ошибки.

Вопросы для обсуждения и задания:

1. Назовите 5 самых главных атрибутов бага.
2. Если у Вас воспроизводится баг, а у разработчика нет и он его закрывает, что делать?
3. Если разработчик говорит, что это не баг и не желает его исправлять, что делать?
4. Приведите пример бага, имеющего высокий Priority, но низкую Severity (помимо примера с ошибкой в названии логотипа компании).
5. Приведите пример бага, имеющего высокую Severity, но низкий Priority.
6. Какие метрики качества Вы знаете?
7. На проекте работают два тестировщика, которые тестируют разные области приложения. Один из них зарепортал 30 багов за день, а другой 5, как можно пояснить эту ситуацию? Стоит ли премировать первого и обратить внимание на работу второго?
8. Если вы сомневаетесь в поведении системы, баг ли это, что будете делать?
9. В пятницу вечером необходимо выйти в Release, то есть выпустить готовую версию продукта, в обед Вы нашли критичный баг. Что делать?

IT-Academy

ТЕМА 1.8 СОЗДАНИЕ ОТЧЕТА О РЕЗУЛЬТАТАХ ТЕСТИРОВАНИЯ

Отчёт о результатах тестирования (Test Result Report, TRR) – часть документации по тестированию, включающая в себя описание процесса тестирования, итоговую информацию о протестированной за подотчетный период функциональности системы, данные о деятельности тестировщиков, планах, возможных проблемах, а также некоторые статистические данные / метрики.

Цель написания TRR – проанализировать текущий процесс на проекте и качество продукта на данном этапе, чтобы выявить возможные проблемы и устранить, а также предоставить лицам, заинтересованным в проекте, данную информацию, которая выражается в конкретных фактах и / или цифрах.

Test Result Report:

- Должен создаваться по расписанию.
 - Заинтересованные лица на проекте должны быть проинформированы о созданном отчете про результаты тестирования.
 - Должен содержать актуальную информацию, не приукрашивая статистические показатели.
 - Составляется ведущим тестировщиком в команде (если Вы – единственный тестировщик на проекте, то это ваша прямая обязанность).
 - Может обсуждаться на митинге (совещании) или носить информативный характер, если согласно отчету, все идет по плану, процессы в команде приводят к поставленным целям и качество продукта на данный момент соответствует ожиданиям.
 - Может быть промежуточным (еженедельным, за спринт, ежемесячным и т.д.), а также финальным, то есть за весь проект.
-
-

Test Result Report состоит из следующих секций (некоторые из них могут отсутствовать для конкретного проекта, так как они могут не нести смысловую нагрузку, так же могут быть добавлены дополнительные секции):

- Команда по тестированию, которая работала над проектом (Testing Team).
 - Описание процесса тестирования (Testing Process Description).
 - Расписание команды (может включать затраченное время), отражающее выполненные задачи (Team Timetable / Schedule).
-
-

- Краткое описание статуса текущего периода (описание процесса и качества) (Summary).
-

- Проблемы, которые мешают запланированному выполнению задачи (Blockers).
-

- Риски (Risks).
-

- Рекомендации (Recommendations).
-

- Список найденных багов (за текущий период и / или за весь период) (Bugs List).
-

- Статистика по ошибкам (распределенная по status, priority, severity и/или модулям приложения) (Bugs Statistics / Overall Bugs Statistics).
-

- Метрики (Metrics) (например, процент “заваленных” билдов; процентное соотношение найденных багов тестировщиками к багам, найденным заказчиком во время UAT; процентное покрытие требований тест-кейсами; процент, показывающий количество пройденных тест-кейсов ко всем созданным тест-кейсам; процент, показывающий количество переоткрытых багов; процентное соотношение требований, на реализацию которых ушло ожидаемое время, к требованиям, на реализацию которых ушло больше запланированного времени).
-

Пример Summary (помните: меньше воды, больше фактов!):

News: Google Chrome Browser v.73 is planned on next Monday, so an additional regression cycle is planned.

Test Status: **Green**

For the reporting week tasks done:

- User Stories analysis – 6
- Created general checklist for 81 idea
- Test-cases writing (45 out of 81):
 - Login module – 8
 - Event creation – 37
- Generated test data:
 - Test email accounts – 12
 - Files for upload – 3
- Reported issues (all medium) – 35

Test Analysis: Although there are 35 bugs found this week, it seems as expected at the current stage of the project, as this build a lot of new functionality was added.

Plans for next week: complete test cases creation, receive build 1.0.1, start test execution.

Problems: Test server is down for 2 days and Help Desk did not reply on our request to restore it yet.

Risks: As far as attempts to fix FIXN-241 has already failed twice, I have to postpone regression test execution, consequently we have a risk for PROD deployment delay.

Заинтересованные в TRR лица:

- Команда тестировщиков в лице ведущего тестировщика.
- Команда разработчиков в лице ведущего разработчика.
- Проектный менеджер.
- Заказчик.
- Могут быть заинтересованы и такие лица, как бизнес-аналитик, архитектор, delivery-менеджер и др. в зависимости от проекта.

Рекомендации и замечания:

- Даже если на проекте не определено расписание, по которому необходимо составлять отчет о результатах тестирования, его нужно определить и следовать этому расписанию, так как не составляя отчет, то есть не анализируя процесс и качество, мы можем пропустить проблемы, не найти вовремя решение и не предугадать серьезные последствия.
- Если заказчику и проектному менеджеру не нужен отчет, то мы его составляем для себя, ведь нам, тестировщикам, он нужен. Только мы можем дать финальное заключение, все ли хорошо происходит на проекте и соответствует ли качество продукта ожиданиям.
- В отчете можно использовать любые средства визуализации информации (таблицы, графики, диаграммы и т.д.), которые помогут быстрее прочитать и понять информацию.
- Отчет должен быть простым для восприятия и понимания.
- Хорошая практика, когда отчеты за все периоды находятся в общем доступе для заинтересованных лиц, что позволяет вернуться к отчету за предыдущий период и проанализировать прогресс.
- TRR может составляться по шаблону заказчика, компании, в которой работает тестировщик или по произвольному шаблону, составленному тестировщиком под конкретный проект в зависимости от требований заказчика, компании и процесса на проекте.

Вопросы для обсуждения и задания:

1. Что такое TRR и для чего нужно его создавать?
2. Если заказчик хочет видеть в TRR только 5 разделов. Какие разделы вы добавите в TRR?
3. Какие метрики качества продукта и / или процесса можете использовать в TRR?
4. Как часто нужно создавать TRR?

ТЕМА 2.1. ОСНОВЫ РАБОТЫ ВЕБ-ПРИЛОЖЕНИЙ

Веб-приложение – это клиент-серверная программа, которая позволяет клиенту (пользователю) получить доступ ресурсам (информации), находящимся на удаленном компьютере (сервере). Чаще всего в качестве клиентского ПО выступает веб браузер.

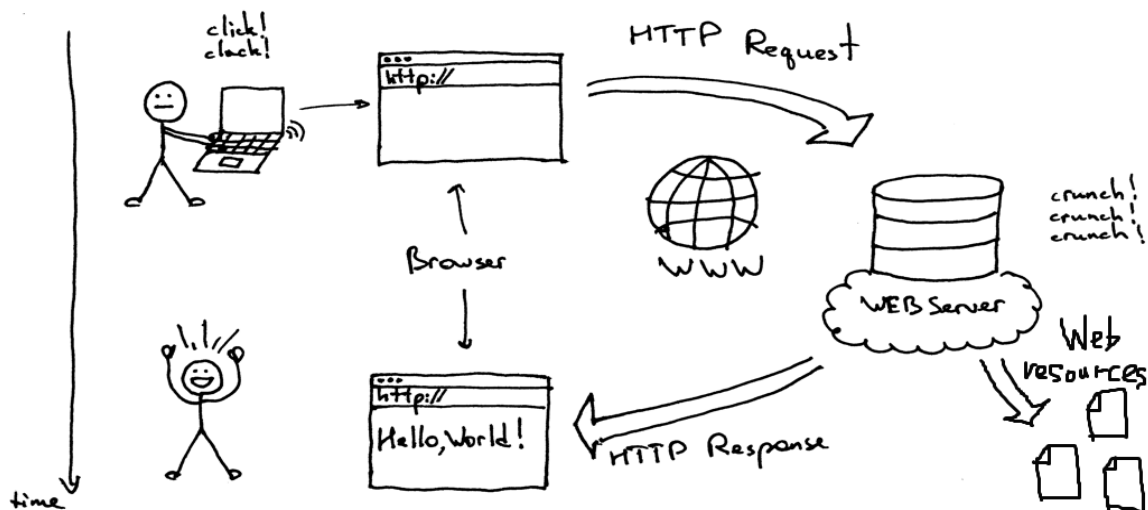
Браузер (веб-обозреватель) – прикладное программное обеспечение, которое используется для:

- Просмотра веб-страниц.
- Просмотра содержания веб-документов.
- Просмотра компьютерных файлов и их каталогов.
- Управления веб-приложениями и других задач.

WEB-сервер (Web-Server) – программное обеспечение, принимающее HTTP-запросы от клиентов, обычно веб-браузеров, и выдающее им HTTP-ответы, как правило, вместе с HTML-страницей, изображением, файлом, медиа-поток или другими данными.

Сервер приложения (Application Server) – программная платформа (удаленный компьютер и вспомогательные программы на нем), предназначенная для установки и исполнения процедур (скриптов, модулей) веб-приложения.

Схематично, работу веб-приложения можно представить следующим образом:



URL – определитель местонахождения ресурса, который является стандартизированным способом записи **адреса** ресурса **в сети Интернет**.

HTTP (Hypertext Transfer Protocol) – протокол прикладного уровня передачи данных между клиентом (веб-браузером) и веб-сервером, который в настоящее время повсеместно используется во Всемирной паутине для получения информации с веб-сайтов.

WEB-сервер является «поставщиком» следующего контента:

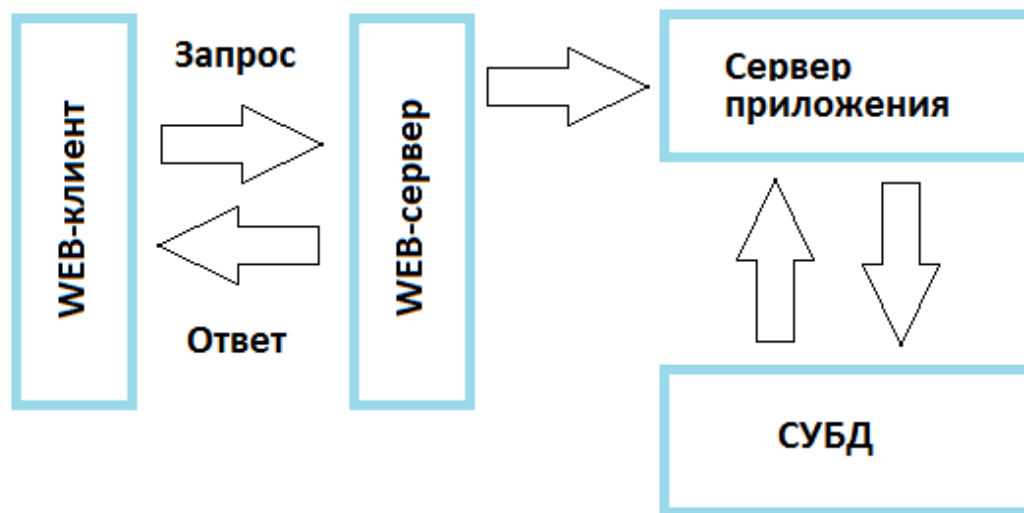
- **HTML** (Hyper Text Markup Language «язык гипертекстовой разметки») – стандартизированный язык разметки документов во Всемирной паутине. Большинство веб-страниц содержат описание разметки на языке HTML. Язык HTML интерпретируется браузерами; полученный в результате интерпретации форматированный текст отображается на экране монитора компьютера или мобильного устройства.
 - **CSS** – формальный язык описания внешнего вида документа, написанного с использованием языка разметки.
 - **JS** – язык программирования, наиболее широкое применение находит в браузерах для придания интерактивности веб-страницам. JavaScript изначально создавался как язык сценариев для того, чтобы сделать web-страницы «живыми». В браузере сценарии подключаются напрямую к HTML и, как только загружается страница, тут же выполняются. Программы на JavaScript – обычный текст. Они не требуют специальной подготовки.
-
-

Контент бывает статический и динамический.

HTTP коды состояния:

- 1xx: Informational – запрос получен, понятен, а обработка продолжается.
 - 2xx: Success – запрос был успешно получен, понятен и обработан.
 - 3xx: Redirection (перенаправление) – для выполнения запроса должны быть предприняты дальнейшие действия.
 - 4xx: Client Error – запрос имеет плохой синтаксис или не может быть выполнен. 404 Not Found – самая распространённая ошибка при пользовании Интернетом, основная причина – ошибка в написании адреса Web-страницы.
 - 5xx: Server Error – сервер не в состоянии выполнить допустимый запрос. 503 Service Unavailable – сервер временно не имеет возможности обрабатывать запросы по техническим причинам (обслуживание, перегрузка и прочее).
-
-

Следующая схема демонстрирует взаимодействие WEB-сервер, Application-сервера, СУБД:



Cookie – небольшой фрагмент данных, отправленный веб-сервером и хранимый на компьютере пользователя. Веб-клиент (обычно веб-браузер) всякий раз при попытке открыть страницу соответствующего сайта пересылает этот фрагмент данных веб-серверу в составе HTTP-запроса. Применяется для сохранения данных на стороне пользователя, на практике обычно используется для:

- аутентификации пользователя;
- хранения персональных предпочтений и настроек пользователя;
- отслеживания состояния сеанса доступа пользователя;
- ведения статистики о пользователях.

Cache – промежуточный буфер с быстрым доступом, содержащий информацию, которая может быть запрошена с наибольшей вероятностью. Доступ к данным в кэше осуществляется быстрее, чем выборка исходных данных из удаленного источника, однако её объём существенно ограничен по сравнению с хранилищем исходных данных.

Вопросы для обсуждения и задания:

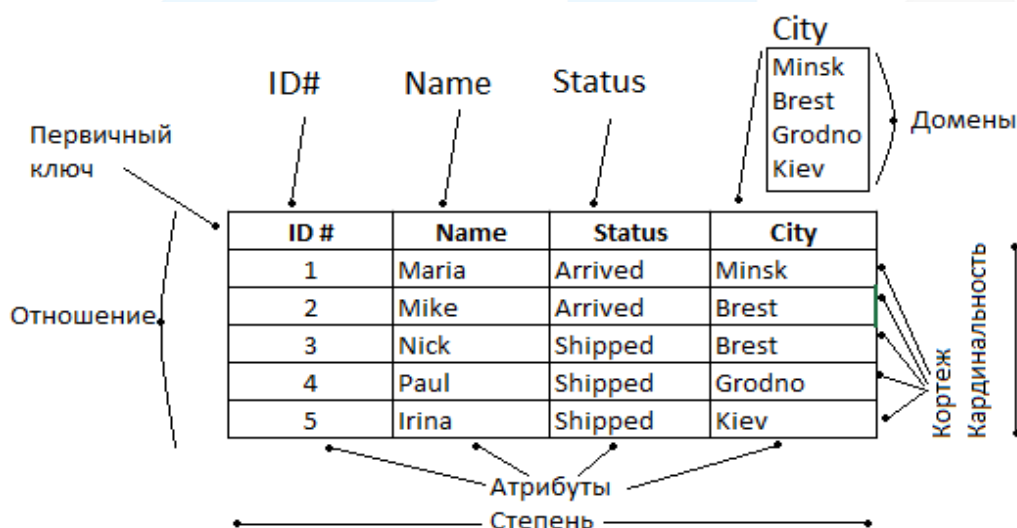
1. Расскажите, что происходит с технической точки зрения, когда пользователь набирает в браузере google.com?
 2. Как в браузере получается веб-страница?
 3. О чем может сигнализировать ошибка 404, 503 на тестовом окружении и какие действия стоит предпринять, если это произошло на проекте, где Вы работаете?
-

ТЕМА 2.2. ОСНОВЫ БАЗ ДАННЫХ И SQL

База данных (БД) – набор сведений, хранящихся некоторым упорядоченным способом.

Система управления базами данных (СУБД) – это совокупность языковых и программных средств, которая позволяет создавать, изменять, удалять данные, осуществляет доступ к ним, обеспечивает безопасность данных и т.д.

Реляционная база данных представляет собой множество двумерных таблиц, логически связанных между собой, каждая из которых содержит информацию об объектах определенного вида.



Первичный ключ (главный ключ, primary key, PK) – представляет собой столбец или совокупность столбцов, значения которых однозначно идентифицируют строки.

Вторичный ключ (внешний ключ, foreign key, FK) – столбец или совокупность столбцов, значения которых должны совпадать со значениями первичного ключа другой таблиц для связи двух таблиц.

Отношения между данными в таблицах:

- Один-к-одному.
- Один-ко-многим.
- Многие-ко-многим.

Нормальные формы – требования, предъявляемые к структуре таблиц в теории реляционных баз данных.

Нормализация – процесс приведения базы данных к виду, соответствующему нормальным формам.

Первая нормальная форма (1NF)

Первая нормальная форма требует, чтобы каждый столбец таблицы имел только одно значение, в каждом столбце было одинаковое количество строк и не было сортировки и дубликатов.

...Ivanov, 15 department, chief...	
------------------------------------	--

Last Name	Position	Department №
Ivanov	Chief	15

Вторая нормальная форма (2NF)

Вторая нормальная форма требует, чтобы таблица находилась в первой нормальной форме и все поля зависели от первичного ключа, а не от его какой-то части.

Ключ может состоять из нескольких полей. Если какие-либо данные в таблице зависят только от одного поля ключа, для нормализации их надо выносить в отдельную таблицу.

Department №	Position	Salary	PC required
15	Chief	10k	yes
15	Engineer	3k	yes
10	Chief	8k	yes
10	Cleaner	1k	no

Третья нормальная форма (3NF)

Таблица находится во второй нормальной форме.

Каждый не ключевой атрибут не транзитивно зависит от первичного ключа.

Employee №	Name	Department	Phone №
1	Ivanov	Functional Department	222-22-22
2	Black	Functional Department	222-22-22
3	Smith	Sales Department	333-33-33

SQL состоит из набора команд, которые позволяют:

- Создавать объекты реляционной базы данных (таблицы).
- Изменять данные в таблицах (вставлять, удалять, исправлять).
- Модифицировать схемы отношений в БД.
- Делать выборки из БД.
- Выполнять вычисления.

Выборка данных:

SELECT ContactName, Address FROM Customers;

SELECT * FROM Customers;

Звездочка (*) на месте списка столбцов обозначает все столбцы таблицы:

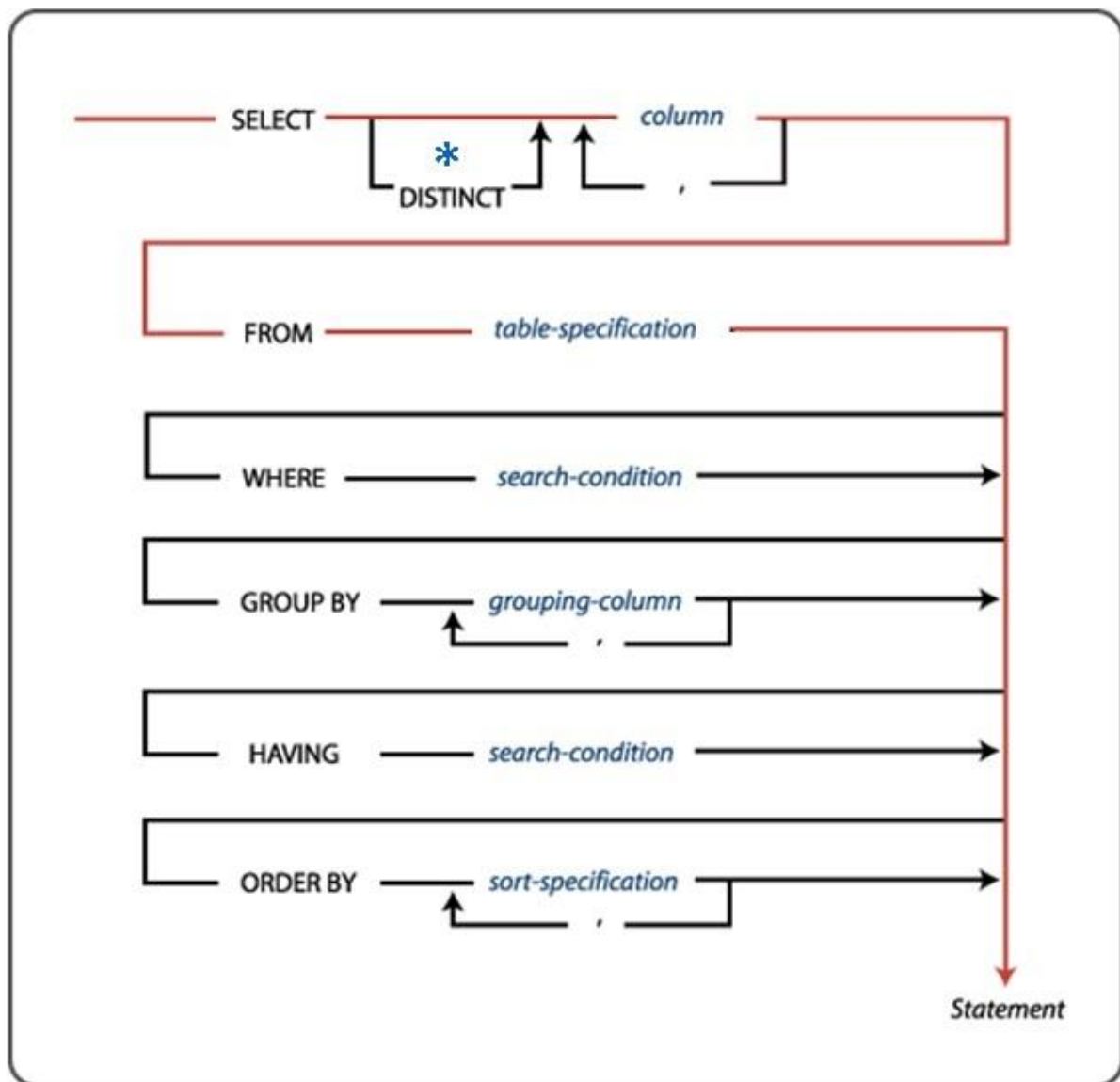
Если в выборке встречаются одинаковые строки, можно от них избавиться, указав ключевое слово **DISTINCT** перед списком столбцов:

SELECT DISTINCT City FROM Customers;

Ключевое слово **LIMIT** помогает выводить не все строки результата запроса, а только, например, первые 5:

SELECT * FROM Customers **LIMIT** 5;

Далее представлена **схема-подсказка** по составлению Select запроса:



WHERE – оператор в SQL, использующийся в качестве условия в SQL-запросе для ограничения записей обрабатываемых или возвращаемых SQL запросом.

В качестве условия (предиката) можно использовать следующие типы выражений: сравнение, диапазон, перечень, текстовую шаблон и поиск пустых значений.

Сравнение:

```
SELECT * FROM Customers WHERE PostalCode = 12209;
```

Диапазон:

```
SELECT * FROM Customers WHERE PostalCode BETWEEN 12200 AND 12210;
```


Перечень:

```
SELECT * FROM Customers WHERE PostalCode IN (12208, 12209, 12210);
```

Текстовый шаблон:

```
SELECT * FROM Customers WHERE PostalCode LIKE '122%':
```

Символ '%' (процент) в шаблоне заменяет собой любую последовательность символов, а символ ' ' (подчеркивание) – один любой символ.

Поиск пустых значений:

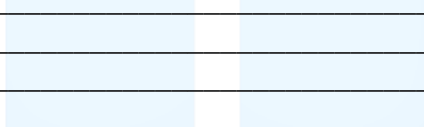
```
SELECT * FROM Customers WHERE PostalCode IS NULL;
```

Условия можно комбинировать про помощи скобок и логических «И» (**AND**) и «ИЛИ» (**OR**), а также ключевого слова **NOT**, которое используется при проверке данных на несоответствие заданному условию:

```
SELECT * FROM Customers
WHERE (PostalCode LIKE '122%' OR PostalCode LIKE '82%') AND PostalCode NOT LIKE
'%23';
```

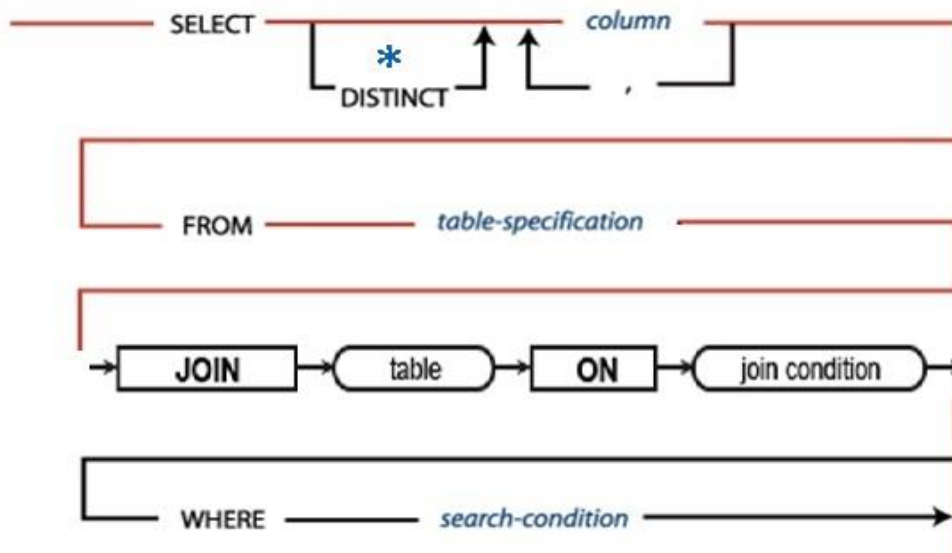
Сортировка данных осуществляется при помощи ключевых слов **ORDER BY:**

```
SELECT DISTINCT City FROM Customers ORDER BY City;  
SELECT DISTINCT City FROM Customers ORDER BY City DESC;
```

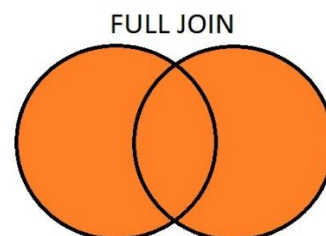
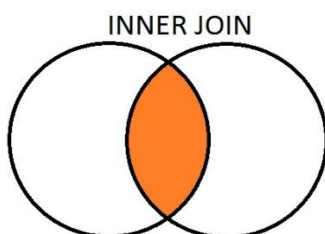
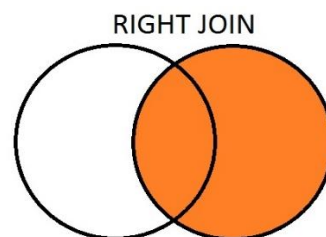
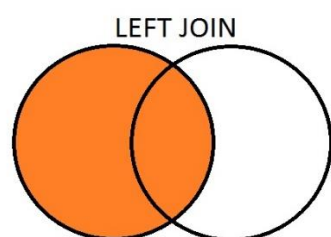


IT-Academy

Логическое связывание таблиц производится с помощью JOIN



```
SELECT Customers.CustomerID, Customers.CustomerName, Orders.OrderID,
Orders.CustomerID, Orders.OrderDate
FROM Customers JOIN Orders
ON Customers.CustomerID = Orders.CustomerID;
```



Оператор **UNION** используется для **объединения результата 2-х и более SELECT** запросов:

```
SELECT Customers.CustomerID, Customers.CustomerName
FROM Customers
UNION
SELECT Orders.OrderID, Orders.CustomerID
FROM Orders;
```

Встроенные функции SQL:

- Агрегатные – вычисляют результирующее значение из ячеек всего столбца.
- Скалярные – преобразуют значения в каждой ячейке.

Некоторые агрегатные функции:

AVG() – возвращает среднее арифметическое значение столбца.
COUNT() – подсчитывает количество строк.
MAX() – вычисление максимального значения.
MIN() – вычисление минимального значения.
SUM() – вычисление суммы значений полей.

Некоторые скалярные функции:

CHAR_LENGTH() – считает количество символов в тексте.
ROUND() – округляет значения.
NOW() – записывает в каждую ячейку текущие дату и время.
FORMAT() – форматирует текст.

Например, можно посчитать всех клиентов:

```
SELECT COUNT(*) FROM Customers;
```

Оператор **GROUP BY** используется вместе с агрегатными функциями **для группировки результата** по признакам одного или более столбцов. **Ключевое слово HAVING** используется в качестве условия в SQL-запросе для ограничения записей, обрабатываемых агрегатными функциями.

Например, можно выбрать всех страны, в которых более 10 клиентов:

```
SELECT Country, COUNT(Country) FROM Customers
GROUP BY Country HAVING COUNT(Country) > 10;
```

Псевдонимы или «alias» используются для присваивания таблице или столбцу временного имени, сокращения имен таблиц и решения проблемы конфликта имен.

```
SELECT Country, COUNT(Country) AS 'Customers Number' FROM Customers  
GROUP BY Country;
```

Примером скалярной функции может быть подсчет количества букв в названиях городов:

```
SELECT City, CHAR_LENGTH(City) FROM Customers;
```

SQL позволяет делать математические и другие операции прямо в процессе выборки данных. Например, выберем имена товаров и их цену, при этом отобразив ее в белорусских рублях и долларах США:

```
SELECT ProductName, CONCAT(Price, ' USD') AS 'USD_Price', CONCAT((Price * 2.07), '  
BYN') AS 'BYN_Price' FROM Products;
```

Иногда невозможно сделать необходимую выборку из имеющихся таблиц, тогда на помощь приходят вложенные запросы. Например, при необходимости выбрать товары с минимальной ценой:

```
SELECT * FROM Products WHERE Price = (SELECT MIN(Price) FROM Products);
```

Базовые команды для начала работы с базой данных:

MYSQL -u user -p	– подключаемся к СУБД (не является SQL командой).
SHOW DATABASES;	– просмотр доступных баз данных.
USE database_name;	– переключение на нужную базу.
SHOW TABLES;	– просмотр списка таблиц в базе.

Создания базы данных:

```
CREATE DATABASE TestDatabase;
```

Создание таблиц:

```
CREATE TABLE clients(  
    id INT(11) NOT NULL PRIMARY KEY,  
    first_name VARCHAR(50) NULL,  
    last_name VARCHAR(50) NULL  
)
```

Добавление новых записей в таблицу:

```
INSERT INTO clients (id, first_name, last_name) VALUES(2, 'Vasil', 'Pupkin');
```

Удаление данных:

```
DROP TABLE имя_таблицы;           – удаление таблицы.  
DROP DATABASE имя_базы;            – удаление базы данных.  
DELETE FROM clients WHERE id = 2;   – удаление записей из таблицы.
```

Обновление данных:

```
UPDATE clients SET first_name = 'Vasily' WHERE last_name = 'Pupkin';
```

Если команды DELETE и UPDATE выполняются без секции WHERE, то соответственно удалены или обновлены будут все строки таблицы. Иначе – только подходящие под заданные условия.

Вопросы для обсуждения и задания:

1. Что такое Foreign Key, приведите пример?
 2. Перед вами запрос с конструкцией LEFT JOIN, можно ли получить такой же результат запроса с помощью RIGHT JOIN, каким образом?
 3. Какие агрегирующие функции знаете?
 4. Существует две таблицы (A и B), в таблице A 10 строк, в таблице B – 7. Сколько строк будет к результирующей таблице, если выполнить left join без условий.
-
-
-
-

ТЕМА 2.3. ПЛАН ТЕСТИРОВАНИЯ. ОСОБЕННОСТИ РАБОТЫ В КОМАНДЕ

План тестирования (Test plan) – документ, описывающий цели, подходы, ресурсы и график запланированных тестовых активностей и определяющий:

- Что тестировать.
- Что не нужно тестировать.
- Кто будет тестировать.
- Где это нужно тестировать и на каком оборудовании.
- Методы и подходы для проектирования тестов.
- Критерии для начала и окончания тестирования, а также любые риски, требующие планирования на случай чрезвычайных обстоятельств.

Ответственным за создание тест-плана обычно является тест лид.

План тестирования создаётся в начале проекта и обновляется по мере необходимости в процессе работы.

Тест план может создаваться по шаблону заказчика, компании, в которой работает тестировщик или по произвольному шаблону.

В общем случае план тестирования **состоит из следующих разделов:**

- **Введение (Introduction)** – краткое описание того, что представляет проект, какая основная цель, кто заказчик, что делает приложение и т.д.

- **Объем работ (Scope of work)** – компоненты и функции, которые должны быть протестированы, компоненты и функции, которые не будут тестироваться, дополнительные сторонние приложения, от которых может зависеть работы или тестирование приложения (лицензии на продукты в том числе).

- **Приемочные критерии (Quality and Acceptance Criteria)** – критерии оценки качества, согласованные с заказчиком, проектным менеджером, для того, чтобы определить, насколько хорошо выполнили задачу и выполнили ли.

- **Факторы, необходимые для успешного завершения проекта (Critical Success Factors)** – правила, которым необходимо следовать, чтобы успешно завершить проект вовремя.

- **Оценка рисков (Risk Assessment)** – список рисков, которые мы имеем на проекте. Описываются вместе с вероятностью возникновения, влиянием на

продукт, предлагаются варианты ликвидации риска или смягчения последствий, или уменьшения вероятности возникновения.

- **Ресурсы (Resources)** – человеческие ресурсы, инструментальные средства, программные средства.
-
-

- **Документация (Test Documentation)** – перечень документов, которые должны быть подготовлены за время или во время работы над проектом, с датами предоставления.
-
-

- **Тестовая стратегия (Test Strategy)** – раздел, описывающий «каким образом, как» будет проходить процесс тестирования, то есть какие подходы, типы, виды, уровни должны применяться.
-
-

- **Расписание (Test Schedule)** – временные рамки процесса тестирования (анализа документации, написания тестовой документации в том числе).
-
-

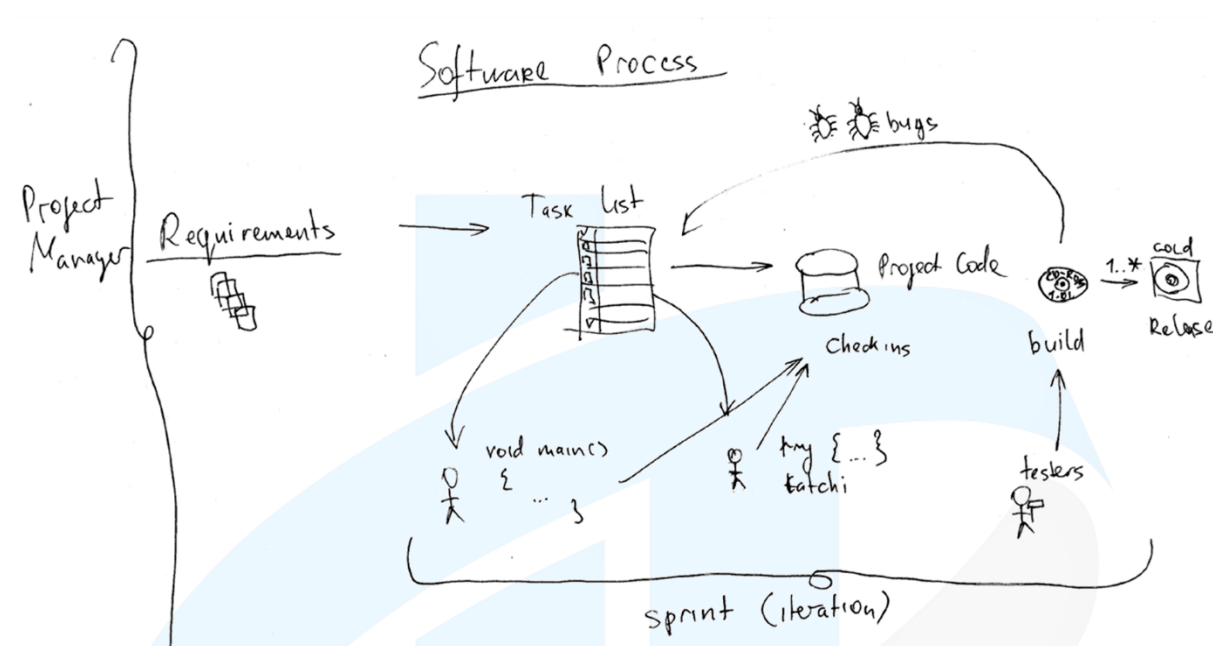
- **Схема тестового окружения (Schema of Test Environment)** – схема тестового окружения.
-
-

Все тест планы уникальны, т.к. тестирование будет разниться в зависимости от заказчика, принятых процессов, команды, оборудования, инструментов, бюджета и многих других факторов.

IT-Academy

Особенности командной работы:

На большинстве проектов по разработке ПО работа строится по схожему принципу:



Команда – группа людей, занятых выполнением определенных задач, необходимых для достижения поставленной цели. Каждый участник группы имеет личную заинтересованность в успехе всей группы. Основная составляющая успешного проекта – команда.

Задачи, как и цели, должны соответствовать правилу **SMART**:

Specific – конкретная.

Measurable – измеримая.

Achievable – достижимая.

Realistic – востребованная.

Timebounded – в конкретных временных рамках.

Для планирования задач руководителю необходимо:

- Оценить объем каждой работы.
- Выстроить цепочку зависимости (одна задача не может быть взята на исполнение, пока другая задача не будет выполнена).
- Выставить приоритет выполнения.
- Понять, как определить, что работа закончена.

Некоторые принципы для успешной работы в команде:

- Делиться знаниями с другими членами команды, особенно если данные знания помогают в достижении цели.
- Доводить до сведения общие цели.
- Планировать время и выполнение задач с учетом задач других членов команды; Например, у тестировщика есть две задачи: протестировать новую функциональность, которую сделал разработчик, и составить тест-кейсы для тестирования функциональности, которую реализовывает разработчик. В такой ситуации стоит отложить написание тест-кейсов и провести тестирование, чтобы разработчик мог исправлять дефекты, когда тестировщик продолжит составление новых тест-кейсов. Иначе может сложиться ситуация, когда разработчик сделал все свои задачи и ждет без работы, когда тестировщик приступит к тестированию и пришлет ошибки на исправление.
- Предупреждать о проблемах, задержках в выполнении задач коллег заранее.
- Проявлять активность на проекте (например, если тестировщик отправил разработчику ошибку на исправление, а он несколько дней не сообщал статус ошибки, то тестировщику можно вежливо уточнить, не пропустил ли он ошибку и какой у него приоритет, план по ее исправлению).
- При планировании отпуска стоит грамотно организовать передачу знаний и задач человеку, который будет замещать во время отпуска, чтобы во время вашего отпуска не оставалось открытых вопросов, о существовании которых знаете только вы или только вы можете решить данный вопрос.

Вопросы для обсуждения и задания:

1. Приведите примеры планов из жизни. Зачем Вы их составляете?
2. Зачем нужен план тестирования на проекте?
3. Назовите 5 основных разделов тест плана
4. Как часто должен обновляться тест план?
5. Что такое SMART цель?
6. Как можно мотивировать сотрудника?
7. Что вы будете делать, если времени не хватает?

ТЕМА 2.4. ТЕСТИРОВАНИЕ ФОРМ, ТЕСТИРОВАНИЕ СОВМЕСТИМОСТИ

В работе с веб-ориентированными приложениями пользователь постоянно встречается с формами (форма регистрации, создания аккаунта, форма покупки и регистрации билета, форма поиска свободных мест на рейс, форма оплаты коммунальных услуг и т.д.).

Формы:

- Пошаговые – новые поля появляются по мере заполнения уже показанных (после загрузки новой страницы или в рамках старой с использованием JavaScript/AJAX).
- Однооконные – все поля (controls) полностью расположены на одной странице.

Основные проверки форм и рекомендации:

После обработки данных формы пользователь должен попадать на некую «информативную страницу».

Местонахождение пользователя должно соответствовать цели визита на данную страницу. Если форма расположена достаточно низко, нужно сделать автоматическую прокрутку к форме.

Для перехода к некоторой операции или следующему шагу операции должно быть достаточно выполнения одного действия. Основные функции приложения, которые вызываются по нажатию на кнопку, должны быть в моментальной доступности для пользователя (например, кнопка «создать письмо» в почтовом ящике является самой крупной, яркой, бросается в глаза, не заставляет пользователя искать).

Если пользователь ввел данные некорректно (или не ввёл вообще), форма должна быть показана заново, и при этом:

- Введенные данные должны сохранять свои значения.
- Следует указать причину неудачи отправки данных для всех полей.
- Неверно заполненные поля визуально выделить.
- Рядом с полем указать суть ошибки и подсказку.

Помещайте рядом с полем или в поле пример его заполнения (placeholder).

Необходимо давать точные и понятные пользователю названия полей, чтобы он не затруднялся при вводе.

Проверяя веб-формы помните, что:

- Пользователь не должен быть телепатом, чтобы пользоваться вашим сайтом.
 - Он не должен повторно выполнять те действия, без которых можно обойтись.
 - Он не должен пытаться понять замысловатые системные сообщения.
 - Он не должен всматриваться в тест, чтобы хоть что-то понять на сайте.
-
-

Проверяйте пошаговые формы на очевидность их функционирования при возврате на предыдущие шаги или восстановлении работы после обрыва связи. Пользователю должно быть ясно, разрешено ли ему возвращаться на предыдущие шаги и механизм переходов должен быть тщательно продуман.

При тестировании формы необходимо уделить внимание не только проверке расположения элементов, правильности названий и красоте. **В первую очередь нужно проверить функциональность формы.**

Например, если форма является формой регистрации, то первым делом стоит проверить, что пользователь зарегистрирован в системе, что данные, введенные пользователем сохранились в базе данных и/или что он может зайти в приложение при помощи введенного имени и пароля. Если форма служит онлайн-оплатой товаров, то необходимо проверить, что количество доступных товаров после покупки уменьшается и определенная сумма списывается с карты оплаты. Если же форма создает какую-либо сущность в системе, то нужно проверить возможными способами, что сущность была создана (найти в базе данных, найти в системе на других страницах приложения и т.п.).

Ниже приведен набор идей/проверок, о которых стоит думать, проверяя форму:

- Расположение формы на экране перед заполнением, после верного заполнения, после неверного заполнения.
- Сохранение/изменение URL при отправке данных из формы.
- Отправка формы с незаполненными обязательными полями.
- Отправка формы с неверно заполненными полями.
- Отправка формы с "нелогичными данными" (дата рождения в будущем).
- Отправка формы с полями, содержащими спецсимволы.
- Восстановление значений полей после отправки формы с неверно заполненными полями.
- Информативность сообщений об ошибках.
- Функциональная сгруппированность полей формы.
- Информативность надписей, подсказок.
- + использовать чек-листы по каждому стандартному полю (текстовое, числовое, дата и т.п.).


Основные поля веб-форм:
Текстовое поле (text field).


Поле пароля (password).


Password


Меню (navbar/navigation bar/main menu/toolbar).

Navbar

 Home

 Link

 Disabled

 Dropdown ▾

Хлебные крошки (breadcrumbs).

[getbootstrap.com](#) / [Documentation](#) / Components

Флажок (check-box).

Checkbox



☒ ☐ ☒ ☐ ☐ ☐

Checkbox Sample

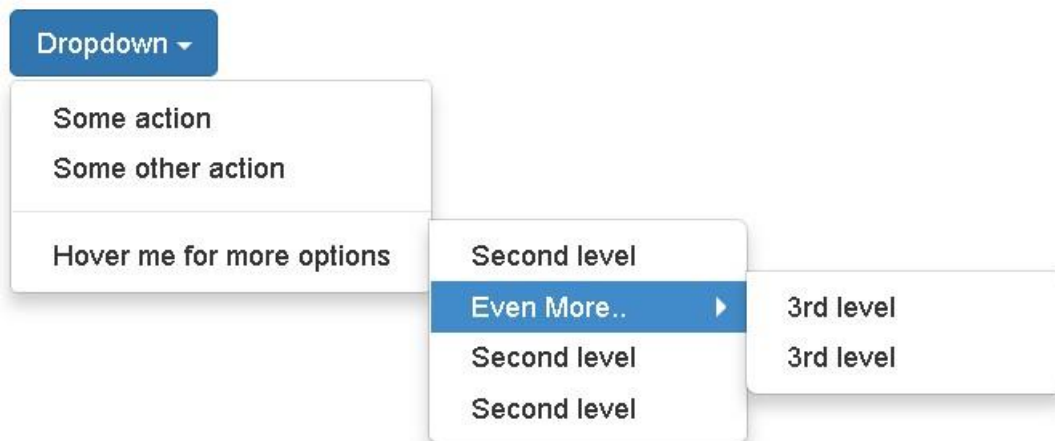
☒ Security

☐ Project

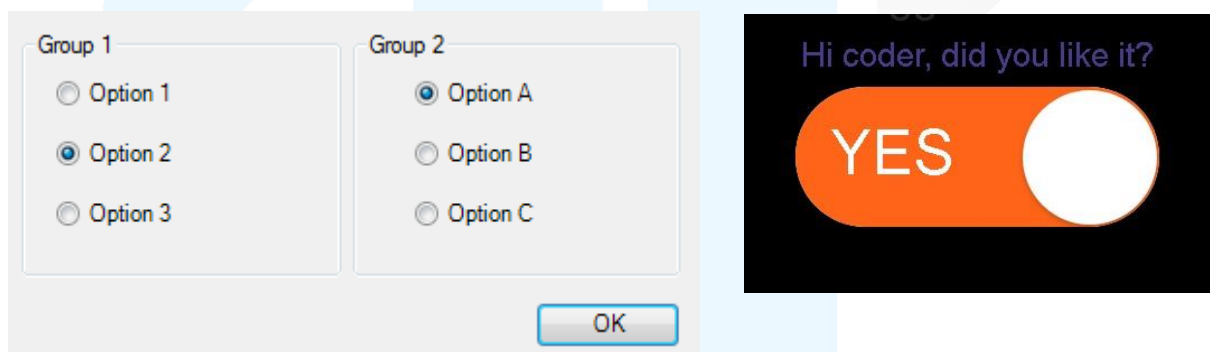
☒ Chart

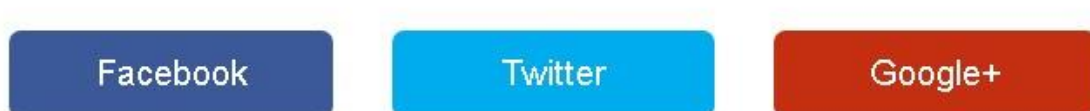
Выпадающий список (dropdown).



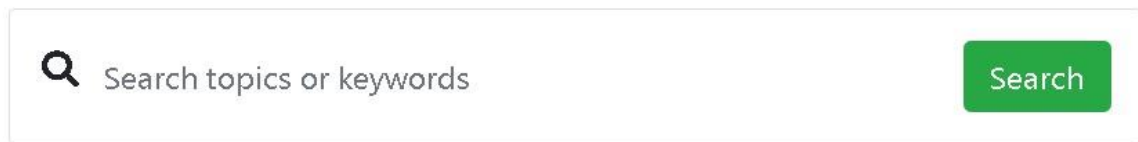
Переключатель (toggle switch, switch, toggle, radio button).



Кнопка (button).

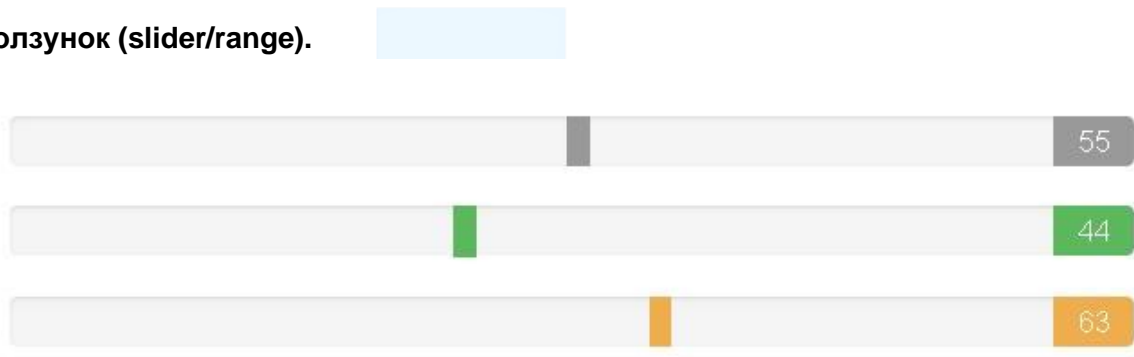


Поиск (search field).



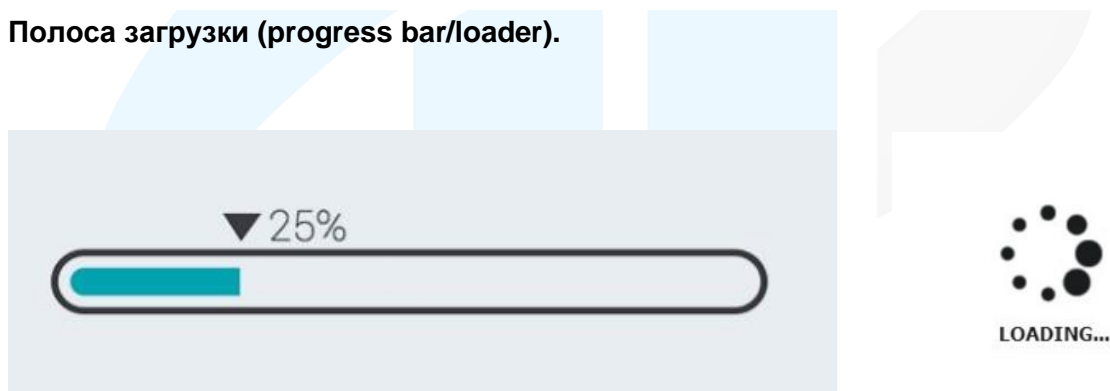
A search field with a magnifying glass icon on the left, the placeholder text "Search topics or keywords" in the center, and a green "Search" button on the right.

Ползунок (slider/range).



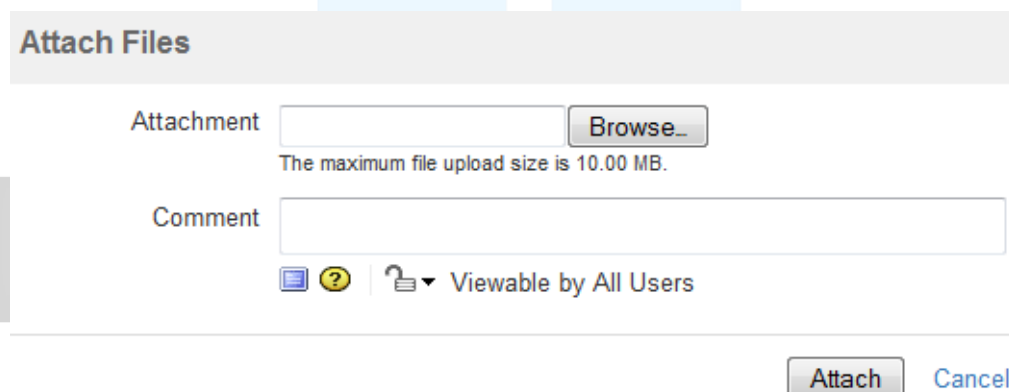
A slider control consisting of three horizontal bars. The top bar has a grey slider and a value of 55. The middle bar has a green slider and a value of 44. The bottom bar has an orange slider and a value of 63.

Полоса загрузки (progress bar/loader).



A progress bar showing 25% completion with a blue fill and a downward arrow icon. To the right is a loading spinner icon with the text "LOADING..." below it.

Поле отправки файла (file upload field)



A file upload form titled "Attach Files". It includes an "Attachment" label, a text input field, and a "Browse..." button. Below the input field is the text "The maximum file upload size is 10.00 MB." There is also a "Comment" label and a text input field. At the bottom right are "Attach" and "Cancel" buttons.

Стандартный чек-лист для проверки **текстового поля**:

- Заполнение поля корректными данными.
 - Оставить поле незаполненным (пустым).
 - Заполнить поле максимально разрешенным количеством символов.
 - Заполнить поле максимально разрешенным количеством символов + 1.
 - Заполнить поле такими спецсимволами как “ ‘ < > \ ; @ и т.п.
 - Заполнить поле только пробелами.
 - Вставить пробелы перед / в середине / в конце текста.
 - Ввести цифры.
 - Ввести данные на разных языках (в т.ч. одновременно).
 - Использовать разные кодировки (cp1251, utf8 и т.п.).
 - Проверить, корректно ли восстановлено значение поля после повторной загрузки страницы (кроме полей с паролями и CAPTCHA).
 - Выравнивание данных в поле ввода.
-
-

Другие поля (стандартный чек-лист)

- Состояние по умолчанию (логично ли оно?).
 - Остаётся ли поле (не)выбранным после повторной загрузки формы.
 - Изменяется ли состояние поля при клике по ассоциированной с ним надписи.
 - «Выбрать все», «Снять отметку со всех», «Инвертировать выбор» (если присутствует).
 - Можно ли кликнуть по кнопке дважды, и какова реакция приложения на такое действие.
 - Можно ли отправить форму, не выбирая ни одну из опций, представленных радиокнопками?
 - Есть ли указание на то, какие файлы (формат, размер) можно отправлять.
 - Наличие фильтра по расширению.
 - Если при отправке файла появляется полоса прогресса, корректно ли она работает.
 - Какова реакция приложения на отправку: несуществующего файла, пустого файла, файла неверного формата, файла с неверным расширением, повреждённого файла, слишком большого файла.
-
-

Советы:

- Добейтесь наличия требований. Тестировать сложную форму без хороших требований крайне тяжело.
- Помните об удобстве и внешнем виде формы.
- Проверьте работу формы под разными браузерами (уточните, в каких необходимо протестировать).
- Загляните в код формы. Изучите его на предмет ошибок и потенциальных уязвимостей.

- Анализируйте, какие данные имеет смысл использовать для тестирования (пример с $2+2$ и $2*2$).
- Представьте, что формой будет пользоваться человек, первый раз в жизни открывший браузер – что он может сделать не так.
- Представьте, что злоумышленник использует форму для атаки на приложение – что он может сделать (PHP, XML, SQL injections)
- Используйте свой опыт и здравый смысл.

Тестирование совместимости (compatibility testing) – тестирование, направленное на проверку способности приложения работать в указанном окружении.

Здесь может проверяться:

- Совместимость с аппаратной платформой, операционной системой и сетевой инфраструктурой (**конфигурационное тестирование, configuration testing**).
- Совместимость с браузерами и их версиями (**кросс-браузерное тестирование, cross-browser testing**) – проверка способности веб-ориентированного приложения идентично работать и отображаться во всех заявленных браузерах. Следует обратить внимание на тот факт, что кроссбраузерная совместимость не обязательно означает «попиксельное соответствие» внешнего вида приложения в разных браузерах.
- Совместимость с мобильными устройствами (**mobile testing**).

Вопросы для обсуждения и задания:

1. Приведите примеры неверного отклика веб-ориентированного приложения на ваш ввод?
2. Можно ли указывать пример пароля в качестве подсказки для заполнения поля?
3. Дайте определения, примеры для следующих понятий: authorization window, drop-down list, radio-button, look up, tab, scroll-bar, link, header, footer, logo, section, side-bar, pop-up, placeholder, hint, edit box.

ТЕМА 2.5. ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА ТЕСТИРОВАНИЯ, ВВЕДЕНИЕ В АВТОМАТИЗИРОВАННОЕ ТЕСТИРОВАНИЕ

Доступ к статическому контенту легко осуществим с помощью **Development Tool (F12)** hot key или **Inspect Element** в контекстном меню выделенного элемента страницы. Development Tool в браузере «Chrome» предоставляет следующие функциональные вкладки:

- Elements.
- Resources.
- Network.
- Sources.
- Timeline.
- Profiles.
- Audits.
- Console.

Console (консоль) служит как для динамической отладки JavaScript, так и для изучения сообщений обо всех проблемах, произошедших при анализе страницы браузером.

Инструмент **Network (сеть)** отображает информацию о всех запросах, выполненных браузером в процессе загрузки страницы (возможно исследование работы с сетью, чтобы определить, какие запросы дольше всего обрабатываются).

Инструмент **Elements** позволяет увидеть HTML страницы (возможно, клиентскую валидацию), CSS-стили элемента Style. Свойства элемента, вычисленные на основе совокупности указанных CSS-стилей (Computed). Можно убедиться в корректности цвета, шрифта и прочего. Вся информация о расположении элемента, отступах и т.п. будет расположена на вкладке Layout.

Все изменения, сделанные в HTML, CSS в Development Tool сразу же будут применимы к странице.

Необходимо проверять, имеется ли валидация правильного ввода данных в поля на стороне WEB-сервера и на стороне сервера приложения.

Например, пользователь не может ввести более двадцати символов в поле, так как на стороне WEB-сервера прописывается ограничение на ввод в виде `maxLength = 20` для input элемента. Если пользователь откроет инструмент разработчика F12 и изменит 20

на 40, то на стороне клиента будет позволено вводить более двадцати символов в поле. Однако при отправке формы на сервер приложения должна появиться валидация со стороны сервера приложения, что ввод более двадцати символов не допустим.

Автоматизация тестирования (Test automation) – набор техник, подходов и инструментальных средств, позволяющий частично исключить человека из выполнения таких задач в процессе тестирования, как управление тестированием, проектирование тестов, выполнение тестов и проверка результатов.

Инструментальное средство автоматизированного тестирования (Test automation tool) – программа (или набор программ), позволяющая создавать, редактировать, отлаживать и выполнять автоматизированные тесты, а также собирать статистику их выполнения.

Характеристики подходящего для автоматизации проекта:

- Долгосрочный проект.
 - Планируются частые запуски тестов.
 - Стабильный функционал.
-
-

Что может быть автоматизировано:

- Проверка ссылок.
 - Тестирование производительности и нагрузочное тестирование.
 - Смоук-тест.
 - Регрессионное тестирование.
 - Конфигурационное тестирование (проверка работоспособности приложения при разных настройках).
 - Часто повторяющиеся тесты, простые для автоматизации.
 - Длинные утомительные для человека тесты, которые возможно автоматизировать.
-
-

Подходящий для автоматизации **тест кейс должен соответствовать следующим характеристикам:**

- Относительно небольшое количество шагов.

- Независимость.
 - Целостность, неделимость.
 - Легко может быть выполнен человеком, который плохо знаком с приложением.
 - Содержит точные входные данные, детальные ожидаемые результаты.
-
-

Преимущества успешной автоматизации:

- Скорость (компьютер работает быстрее человека; время – деньги).
- Надёжность (снижен риск ошибок по причине человеческого фактора).
- Мощность (можно выполнить действия, недоступные человеку).
- Средства автоматизации тестирования собирают информацию и представляют её в числовой, графической форме, удобной для понимания человеку.

Недостатки автоматизированного тестирования:

- Необходим серьёзный анализ и планирование экономической целесообразности автоматизированного тестирования с учетом плана развития проекта.
- Не все области приложения можно протестировать с помощью автоматизированных тестов.
- Необходима грамотная стратегия разработки и управления тестами, тестовыми данными и т.п.
- Необходим квалифицированный персонал с сильной технической базой.
- Необходимы специальные инструментальные средства, которые могут быть очень дорогими.
- В случае серьёзных изменений в приложении многие автоматизированные тесты становятся бесполезными и/или требуют значительной переработки.

Record&Playback («Записать и воспроизвести») – технология, позволяющая выполнить с тестируемым приложением некоторый набор действий, которые будут записаны на специальном языке программирования, а затем могут быть воспроизведены данным средством.

Базовый простой инструмент для начинающих в автоматизированном тестировании – Selenium IDE/Katalon recorder. Во время записи каждый шаг имеет ключевые поля:

- command
 - target
 - value.
-
-
-

Вопросы для обсуждения и задания:

1. Приведите примеры, в каких ситуациях могут пригодиться знания development tool?
2. Вы единственный тестировщик на проекте. Заказчик предлагает добавить автоматизированное тестирование в тестовую стратегию. Поддержите предложение? Как будете рассуждать?
3. Проанализировав проект вы пришли к выводу, что автоматизация будет целесообразна. Что отдадите в автоматизацию в первую очередь? Что потом?
4. Какие основные поля каждого шага теста в Selenium IDE?

IT-Academy

ТЕМА 2.6. ДОМЕННОЕ ТЕСТИРОВАНИЕ И ДРУГИЕ МЕТОДЫ ПРОЕКТИРОВАНИЯ ТЕСТОВ

Доменное тестирование (Domain testing) – вид тестирования (проектирования тестов), направленный на анализ показательных значений и взаимосвязи элементов. Оно включает в себя Эквивалентное разделение и Анализ граничных значений (boundary value analysis).

Эквивалентное разбиение (equivalent partitioning) – разработка тестов методом, в котором тестовые сценарии создаются для проверки элементов эквивалентной области.

Класс эквивалентности – набор тестов, полное выполнение которого является избыточным и не приводит к обнаружению новых дефектов. Если мы ожидаем одинакового результата от выполнения двух и более тестов, эти тесты эквивалентны.

Признаки эквивалентности. Несколько тестов эквивалентны, если:

- Они направлены на поиск одной и той же ошибки.
 - Один из тестов обнаруживает ошибку, другие её тоже, скорее всего, обнаружат.
 - Один из тестов НЕ обнаруживает ошибку, другие скорее всего, НЕ обнаружат.
 - Тесты используют схожие наборы входных данных.
 - Для выполнения тестов мы совершаем одни и те же операции.
 - Тесты генерируют одинаковые выходные данные или приводят приложение в одно и то же состояние.
-
-

Граничные условия (border conditions) – это те места, в которых один класс эквивалентности переходит в другой. Граничные условия очень важны, их обязательно следует проверять в тестах, т.к. именно в этом месте чаще всего и обнаруживаются ошибки.

Классы эквивалентности и/или граничные значения можно выделить в таких сущностях, как числа, символы, количество (записей в БД, строк), длина строки, размер файла, объем памяти, разрешение экрана, версии операционной системы, библиотек.

В общем подход можно описать так:

- Определить набор функций.
 - Определить переменные.
 - Разделить пространство значений на группы.
 - Выбрать «наиболее показательные» значения, представляющие каждую группу, включая границы.
 - Провести тесты с их использованием.
-
-

Например:

Upload a new file *

Choose File

No file chosen

Upload

Files must be less than 2 MB.

Allowed file types: png gif jpg jpeg.

Примеры классов эквивалентности для некоторых сущностей:

«Измерение»	Классы эквивалентности	Примеры значений
Длина	внутри диапазона	50
	границы	1, 1000
	за границами	0, 1001
Символы	алфавитные	abc
	цифровые	123
	спецсимволы	!@#\$%^&*()_+ =\\{}[]:~';'<>?.,/
	разные языки	мама мыла раму
	разные кодировки	ÀÇÈÌÑÒÙẀàçèìñò
	специальные тесты	sql, xss
Пробелы	нет пробелов	aaaaaaaa
	пробелы в начале, в конце	_a, _a_, a_
	пробелы в середине	a_a
Уникальность, Регистр	разные значения	aaa / bbb
	одинаковые значения	aaa / aaa
	различия в регистре	aaa / aAa
Способ заполнения	набор с клавиатуры	набрать текст
	вставка	вставить текст

Помните:

- Несколько позитивных значений можно проверить за раз.
- При этом негативные значения так комбинировать нельзя, т.к. мы должны убедиться, что программа корректно отслеживает проблемы с каждым из них.
- Вводим в поле, не принимающее спецсимволы, полный набор таковых. Если что-то пошло не так, вводим половину, потом половину половины и т.д., чтобы найти на какие из них возникает ошибка.
- Если надо выяснить максимальную длину принимаемого текста, каждый раз удваиваем её. Когда нашли проблему, добавляем по ¼ от предыдущего успешного результата.

Тестирование всех пар (All Pair Testing) – это техника составления тестов, основанная на том предположении, что большинство дефектов возникает при взаимодействии не более двух параметров. Тестовые наборы, генерируемые при использовании данной методики, охватывают все уникальные пары комбинаций параметров, что считается достаточным для обнаружения большего числа дефектов.

Параметры:

☒ Check box

Lists

Text Field

☒ Radio

☐ Buttons

Значения параметров:

Drop-down list

Option 1

Option 2

Option 3

Option 4

Option 5

Option 6

Lists -> Option 1

Lists -> Option 2

Lists -> Option 3

Lists -> Option 4

Lists -> Option 5

Lists -> Option 6

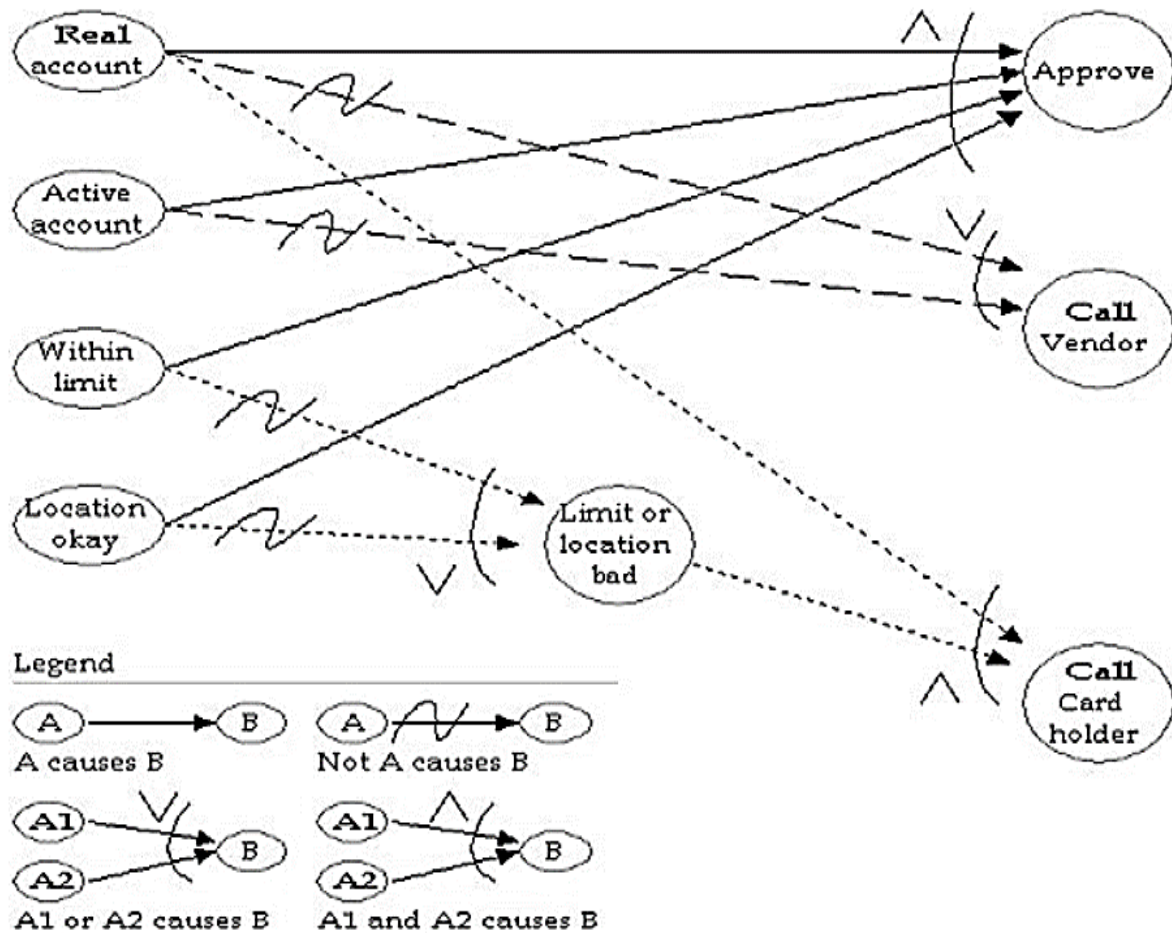
При составлении наборов тестов, в которых учитываются все пары значений параметров, используется инструмент Pict (также существуют другие онлайн инструменты), который генерирует тестовые наборы, исключая полный перебор.

Тестирование по сценариям использования (use case testing, UC testing) – разработка тестов методом черного ящика, при котором тестовые сценарии создаются на основе сценариев использования (UC, Use Case).

Диаграмма причинно-следственных связей (диаграмма Исикавы или диаграмма «рыбьей кости» (Fishbone Diagram), или «причинно-следственная» диаграмма (Cause and Effect Diagram)) – это графический метод представления и анализа причинно-следственных связей, проблем или сущностей.

Таблица принятия решений (таблица решений, Decision tables) – это способ представления связи между множеством независимых условий и действий в виде таблицы.

Пример графа причинно-следственных связей:

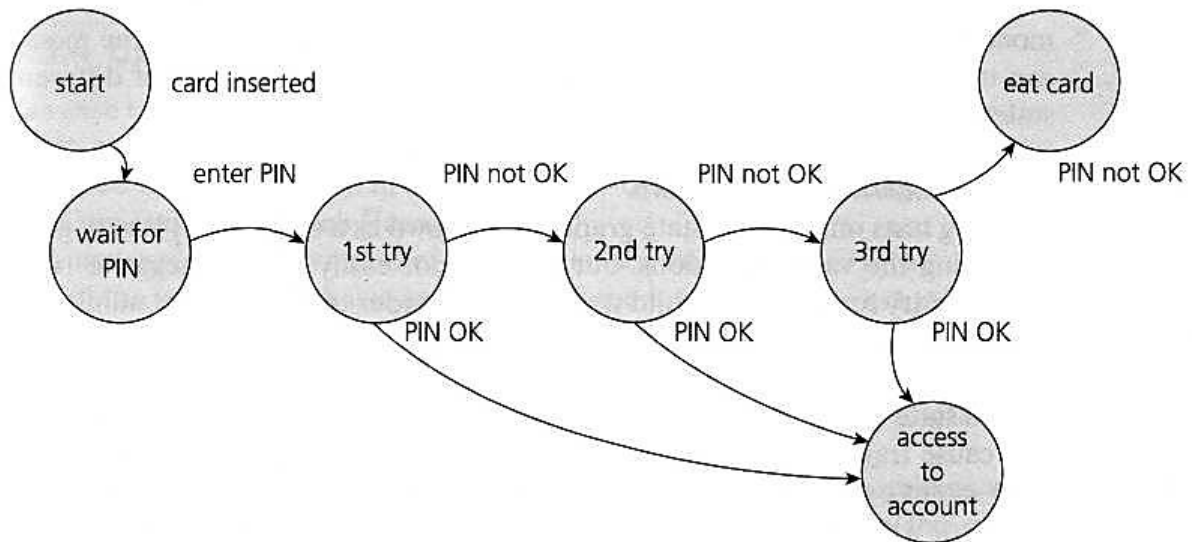


Пример таблицы решений:

Less than 50 Units ordered	Y	Y	Y	Y	N	N	N	N
Cash on Delivery	Y	Y	N	N	Y	Y	N	N
Wholesale Outlet	Y	N	Y	N	Y	N	Y	N
Discount Rate 0%				X				
Discount Rate 2%		X	X					X
Discount Rate 4%	X					X	X	
Discount Rate 6%					X			

Диаграмма переходных состояний (State-Transition diagram) – графический метод представления различных состояния приложения, показывающий, как эти состояния могут меняться под воздействием событий, которые возникают во время работы приложения, а также реакцию приложения на эти события.

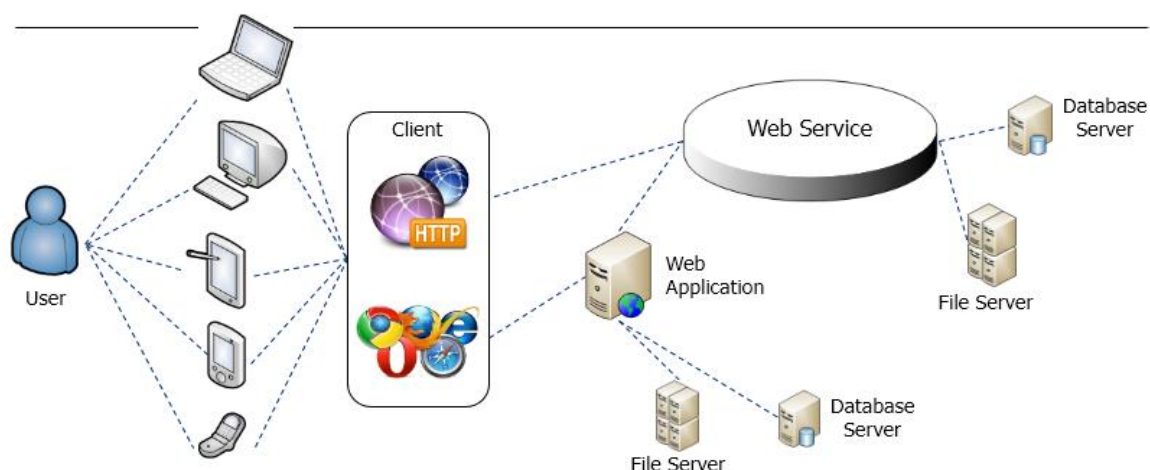
Таблицы переходных состояний (State-Transition table) – это способ представления всех возможных вариантов переходов состояний системы в виде таблицы.



Вопросы для обсуждения и задания:

1. Что такое классы эквивалентности и граничные значения?
2. Как можно протестировать большое количество комбинаций входных данных?
3. Какой метод проектирования тестов подходит для приложения со сложной логикой?
4. Какие методы черного ящика вам известны?

ТЕМА 2.7. ОСНОВЫ ТЕСТИРОВАНИЯ ВЕБ-СЕРВИСОВ REST, ФОРМАТ ОБМЕНА ДАННЫМИ JSON



WEB-Service (WS) – это веб-ориентированная технология, которая позволяет программам общаться между собой, используя стандартные форматы (XML, JSON) и посредством специальных протоколов (SOAP, REST) в сети Интернет.

Отличительные особенности	
Web Application	Web Service
Информация понятна человеку	Используется программами
Есть графический интерфейс	Не имеет графического интерфейса

Примеры веб-сервисов:

- Поисковая система.
- Система, предоставляющая прогноз погоды.
- Система, предоставляющая курсы валют.

На данный момент наибольшее распространение получили следующие протоколы (правила, стандарты) реализации веб-сервисов:

- SOAP (Simple Object Access Protocol).
- REST (Representational State Transfer).
- XML-RPC (XML Remote Procedure Call).

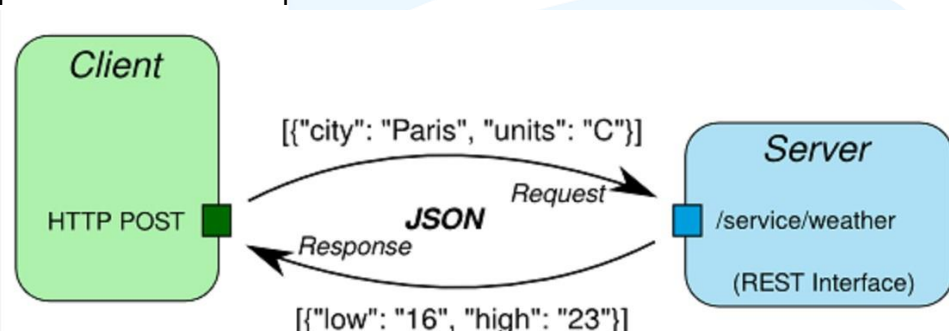
SOAP (от Simple Object Access Protocol – простой протокол доступа к объектам) – протокол обмена структурированными сообщениями в формате XML между программами.

REST – архитектурный стиль (не протокол) проектирования и взаимодействия компонентов распределенного приложения в сети Интернет.

Форматы обмена данными:

- Можно использовать для обмена информацией между двумя несовместимыми системами.
- Используются для хранения данных в файловой системе так же, как и для сохранения и выборки информации в базе данных.
- Служат для описания данных.

Пример работы REST веб-сервиса:



JSON – текстовый формат обмена данными, основанный на JavaScript. С одной стороны, JSON легко читается людьми, с другой – может использоваться практически с любым языком программирования.

JSON-формат представляет собой (в закодированном виде) одну из двух структур:

- Набор пар **ключ:значение**.
- Упорядоченный **набор значений**.

Объект – это неупорядоченное множество пар **ключ:значение**, заключенное в фигурные скобки «{ }». Между именем и значением стоит символ двоеточие «:», а пары ключ-значение отделяются друг от друга запятыми «,».

Пример: {"name": "Петр"}

Ключ описывается строкой.

В качестве **значений** в JSON могут быть использованы строка в двойных кавычках, число, логическое значение (true или false), объект, массив или значение null.

Строка – это упорядоченное множество из нуля или более символов юникода, заключенное в двойные кавычки.

Число – просто число, заключать в двойные кавычки не нужно.

Например, в качестве значений "postalCode": в JSON могут быть использованы как число "postalCode": 101101 так и строка "postalCode": "101101"

Массив (одномерный) – это множество объектов. Массив заключается в квадратные скобки []. Значения отделяются запятыми.

Пример:

```
{{"name": "Петя"}, {"name": "Вася"}, {"name": "Миша"}]  
"phoneNumbers": ["812 123-1234", "916 123-4567"]
```

JSON Schema – один из языков описания структуры JSON-документа. Использует синтаксис JSON. Базирован на концепциях XML Schema. JSON Schema – самоописательный язык: при его использовании для обработки данных и описания их допустимости могут использоваться одни и те же инструменты сериализации/десериализации.

Тестирование производительности (performance testing) – анализ «скоростных показателей» приложения при различной нагрузке.

Пример: как будет работать сайт при росте количества посещений.

Тестирование производительности включает в себя следующие виды тестирования:

Нагрузочное тестирование (load testing) – исследование способности приложения сохранять заданные качественные показатели при нагрузке в допустимых пределах и в некотором их превышении (запас прочности приложения).

Пример: сможет ли сайт удерживать среднее время отклика в пределах 500 мс при росте посетителей до 2000.

Стресс-тестирование (stress testing) – анализ поведения приложения при неспецифических (неожиданных) изменениях нагрузки (в штатных условиях).

Пример: как поведет себя сайт в случае 10000 одновременных посещений.

Объёмное тестирование (volume testing) – исследование производительности приложения при обработке различных объёмов данных.

Пример: смогут ли 2000 пользователей загружать файлы в галерею сайта.

Цели тестирования производительности:

- Оценка максимальной и минимальной производительности системы.
 - Определение лучшей архитектуры системы, выбор наилучшей платформы.
 - Определение узких мест системы.
 - Оценка и оптимизация схемы БД.
 - Определение максимального числа одновременно работающих пользователей.
 - Определение оптимального способа хранения файлов.
 - Определение влияния конфигурации системы на производительность.
 - Оценка показателей масштабируемости системы.
-

Зачем проводить тестирование производительности:

- Плохая производительность раздражает пользователя, он может уйти.
 - Плохая производительность может приводить к проблемам с безопасностью.
 - Плохая производительность уменьшает количество операций в единицу времени, выполняемых системой, что приносит убытки заказчику.
-

Характеристики производительности системы и тесты для их исследования:

- **Время отклика** – позволяет оценить время отклика (response time and latency) системы.
 - **Мощность** – тест на выживаемость (longevity test) – показывает способность системы работать длительное время под высокой нагрузкой.
 - **Стабильность** – тест «часа пик» (rush hour test) – позволяет оценить реакцию системы на резкое изменение нагрузки.
 - **Масштабируемость** – наращивание нагрузки (ramp up) – позволяет определить «точку насыщения» (saturation point) – показатель нагрузки, при которой производительность системы начинает ухудшаться.
-

Наиболее известные инструментальные средства тестирования производительности:

- Apache Jmeter.
 - Grinder.
 - HP Performance Center (+ HP LoadRunner).
 - Rational Performance Tester.
 - SilkPerformer.
 - TestComplete.
-

JMeter – инструмент для проведения нагрузочного тестирования, разрабатываемый Apache Jakarta Project.

- Имеется возможность создания большого количества запросов с помощью нескольких компьютеров при управлении этим процессом с одного из них.
- В программе реализованы механизмы авторизации виртуальных пользователей, поддерживаются пользовательские сеансы (сессии).
- Организовано протоколирование результатов теста и разнообразная визуализация результатов в виде диаграмм, таблиц и т. п.
- Способен проводить нагрузочные тесты для JDBC-соединений, FTP, LDAP, SOAP, JMS, POP3, IMAP, HTTP и TCP.
- Архитектура, поддерживающая плагины сторонних разработчиков, позволяет дополнять инструмент новыми функциями.

JMeter позволяет записывать последовательность шагов пользователя в приложении с применением технологии Record & Playback. Для этого в JMeter необходимо включить прокси-сервер: в меню слева выберите «Workbench» -> «Edit» -> «Add» -> «Non-Test Elements» -> «HTTP Proxy Server». Нажмите «Start» внизу появившейся страницы.

После того, как прокси-сервер (proxy-server) был создан, сконфигурирован и запущен вам нужно сконфигурировать прокси-сервер в браузере (если вы не меняли параметры по умолчанию, то это будет: адрес «127.0.0.1» порт «8888»). В случае Mozilla Firefox можно использовать дополнение ProxySwitcher Tool для более удобного переключения его работы.

Далее выполните вручную все шаги теста для дальнейшего его автоматизирования, JMeter их записывает.

После того, как запись теста завершена, необходимо остановить прокси-сервер JMeter и вернуть настройки браузера в состояние, в котором они находились до того, как браузер был перенастроен на использование прокси JMeter. В тесте можно удалить ненужные действия.

На следующем этапе выберите в левом меню «Test plan» -> «Edit» -> «Add» -> «Thread users») и создайте, настройте группу пользователей. Далее добавляйте в тестовый план инструмент сбора статистики («Edit» -> «Add» -> «Listeners»).

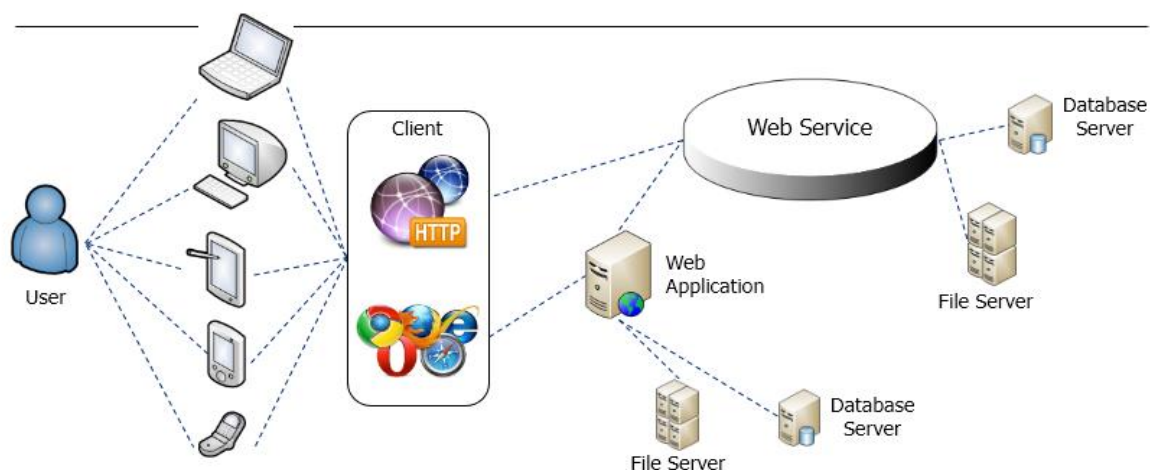
Перед тем, как запустить тест, переместите элементы теста в дочерние элементы созданной группы пользователей. И тест можно запускать («Run» -> «Start»). Во время выполнения теста в правом верхнем углу окна JMeter будет отображаться зелёный квадрат и количество активных виртуальных пользователей.

Вопросы для обсуждения и задания:

1. Что такое JSON и для чего он используется?
2. Приведите письменно пример объекта в формате JSON?
3. Опишите помещение, в котором вы находитесь в формате JSON.
4. Что такое веб-сервис и приведите пример использования?
5. Какие типы веб-сервисов вы знаете?
6. Дайте определение понятиям performance, load, stress, volume testing.
7. Зачем проводить тестирование производительности, какие у него цели?

IT-Academy

ТЕМА 2.8. ОСОБЕННОСТИ ТЕСТИРОВАНИЯ ВЕБ-СЕРВИСОВ, ФОРМАТ ОБМЕНА ДАННЫМИ XML



SOAP (Simple Object Access Protocol) – всегда один вход (ссылка) на сервис, а указание какую процедуру или функцию вызывать для обработки поступающих данных мы передаем вместе с данными.

WSDL (Web Services Description Language) – язык описания веб-сервисов и доступа к ним, основанный на языке XML. Описывает сообщения, заголовки, события, которые свойственны для веб-сервиса.

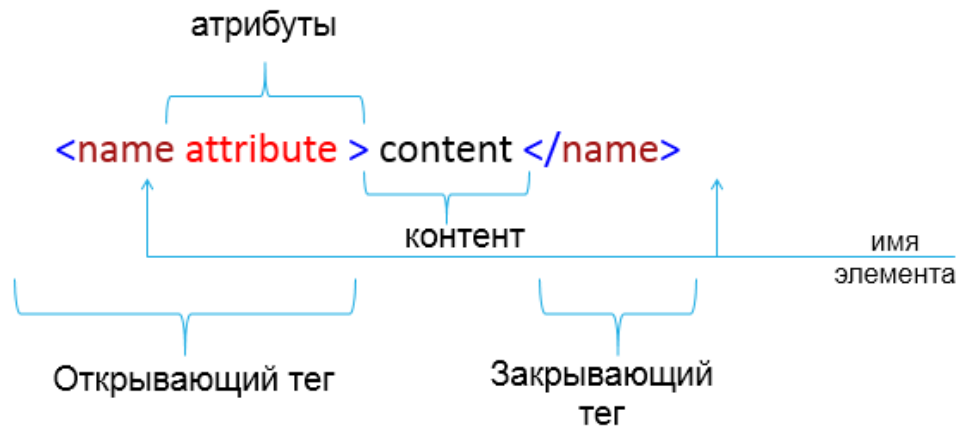
XSD – язык описания структуры XML-документа и типы данных, которые там могут храниться. Имея XSD документ, можно составить XML документ, который будет правильно интерпретирован веб-сервисом.

XML – расширяемый язык разметки, используемый для хранения и передачи данных.

Особенности/недостатки XML формата:

- XML похож на HTML. Тэги в HTML заданы жестко, XML тэги надо определять.
- Для описания правил XML данных используются XML Schema.
- XML можно использовать для обмена информацией.
- XML используется для хранения данных в файловой системе.
- XML обычно менее лаконичен, чем аналогичные форматы (например, JSON).
- Передача XML создает большой трафик / загружает процессор.
- Обработка XML может быть медленнее и более требовательна к памяти.

Структура XML элемента:



Элементы обозначают именованные разделы информации. Элемент может содержать: другие элементы, текст, атрибуты или набор из всего выше названного.

Теги задают границы элемента. Открывающие теги обозначают начало элемента.

```
<elementName att1Name="att1Value"
              att2Name="att2Value" ...>
```

Закрывающие теги обозначают конец элемента. Они не могут иметь атрибутов.

```
</elementName>
```

Пустые теги используются для создания элементов не имеющих текстового содержания. Могут включать атрибуты.

```
<elementName att1Name="att1Value" />
```

Если при определении элементов необходимо задать какие-либо параметры, уточняющие его характеристики, то имеется возможность использовать атрибуты элемента.

```
<color RGB="true">#ff08ff</color>
<color RGB="false">white</color>
<font color="white"
      name="Arial">Black</font>
```

Все элементы должны иметь закрывающие теги:

```
<?xml version="1.0" encoding="UTF-8"?>
<person>
  <familyName>Kress</familyName>
  <not_closed_element>
</person>
```

XML документ может иметь только один корневой элемент:

```
Валидный XML
<?xml version="1.0" encoding="UTF-8"?>
<person>
  <givenName>Peter</givenName>
  <familyName>Kress</familyName>
</person>
Не валидный XML
<person>
  <givenName>John</givenName>
  <familyName>Doe</familyName>
</person>
<person/>
```

Названия элементов являются регистр зависимыми, поэтому и открывающий и закрывающий тег должны быть в одном регистре:

```
<message>This is correct</message>
<Message>This is incorrect</message>
```

Элементы не могут пересекаться:

```
Валидный XML
<person>
  <givenName>Peter</givenName>
  <familyName>Kress</familyName>
  <nickName>KrePet</nickName>
</person>
Не валидный XML
<person>
  <givenName>John</givenName>
  <familyName>Doe<nickName>KrePet</familyName></nickName>
</person>
```

Все значения атрибутов должны быть заключены в кавычки:

```
Валидный XML
<person name="John" surname='Doe' />
Не валидный XML
<person name=John/>
```

<, >, & не могут использоваться в текстовых блоках:

```
Валидный XML
<text>I &my dog</text>
Не валидный XML
<text>I & my dog</text>
```

Особенности тестирования WS:

- Нет GUI.
 - Нужны специальные инструменты для выполнения тестов (SOAP UI, REST Client, Postman, Fiddler).
 - Дополнительные требования могут потребовать специальных знаний (XSD, WSDL, SOAP).
 - Входные данные должны иметь определенный формат (JSON, XML).
-
-

Важно выяснить:

- Какие методы поддерживает WS?
 - Какие параметры/атрибуты у методов?
 - Образцы запросов (можем взять у разработчика)?
 - Определить набор тестов для каждого метода (позитивные, негативные с учетом классов эквивалентности и граничных значений и т.п.).
 - Разделить тесты на группы (функциональность/приоритет).
-
-

Специфические тесты для WS:

- Пустые элементы и элементы, содержащие null – `<empty/>` and `<empty></empty>` .
 - Комментарии `<!-- Insert comments in all requests -->` .
 - Проверить правильность («валидность») ответов согласно схеме (если она есть).
 - Проверить с обязательными атрибутами, без обязательных, с дополнительными, с недопустимыми.
 - Различные типы данных (строка вместо числа и т.п.).
 - Дубликаты атрибутов/элементов.
 - Различный порядок атрибутов/элементов.
 - Проверка длины строк, величины чисел.
 - Неверный формат данных (нарушение синтаксиса JSON, XML).
-
-
-

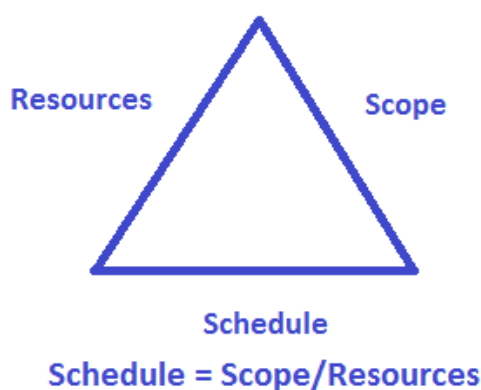
Вопросы для обсуждения и задания:

1. Что такое XML и для чего он используется?
 2. В чем отличие HTML от XML?
 3. Опишите помещение, в котором вы находитесь в формате JSON.
 4. В чем отличие SOAP от REST?
 5. Почему в HTML теги придумывать самому нельзя, а в XML можно?
-
-
-

ТЕМА 3.1 ОЦЕНКА ТРУДОЗАТРАТ ПРИ ТЕСТИРОВАНИИ

Необходимо **планирование времени** на:

- Анализ документации.
- Создание тестовой документации.
- Прохождение тестов.
- Создание отчета о результатах тестирования.
- И т.д.



Методы эстимирования (могут интегрировано использоваться во время планирования, что повышает точность определения времени или создает базу для аналитики опыта прошлых проектов):

- By Test Cases (Основываясь на тест-кейсы).
- Based on development (Основываясь на время разработки).
- Аналогии с другими проектами и рекомендации экспертов.
- Декомпозиция.
- Эстимирование по 3-м точкам;
- и многие другие.

By Test Cases

- Довольно точна – абстрактно, но очень устраивает заказчиков.
- Проста для начинающих.
- Прозрачна (никакой мистики).
- Широко применяется.
- Проверена: работает на множестве проектов.

Как посчитать:

- Разделить задачу на фрагменты (можно использовать чек-лист).
 - Подсчитать число тестов.
 - Учесть создание и число проходов.
 - Учесть свою производительность (минут на тест, тестов в день).
 - Учесть другие задачи (bug reporting, verification troubleshooting, reports, emails, meetings).
 - Риски (illness, build failed, some tests not counted, development delays, etc.).
-
-
-

Based on development

Необходимо взять за основу время, запланированное на активности разработки, посчитать на его основе время на тестирование по правилу – ~ 30% от времени разработки.

Важно: не стоит бездумно использовать данный метод, так как бывают обратные случаи, когда время на тестирование составляет 250% от времени разработки.

Эстимирование по 3-м точкам

- Оптимистическое время (optimistic time) (O): минимально возможное время выполнения задачи, в предположении, что все происходит наилучшим образом.
 - Пессимистическое время (pessimistic time) (P): максимально возможное время, требующееся для выполнения задачи, в предположении, что все происходит наихудшим образом (исключая крупные катастрофы).
 - Наиболее вероятное время (most likely time) (M): оценка времени, требующегося для выполнения задачи, в предположении, что все происходит как обычно.
 - Ожидаемое время (expected time) (ET): лучшая оценка времени, требуемого для выполнения задачи, учитывая, что вещи не всегда происходят как обычно. (Ожидаемое среднее время выполнения задачи, если она будет повторяться многократно).
 - $ET = (O + 4M + P)/6$
-
-
-

Рекомендации:

- Если достаточно информации, требования ясны, то эстимируем (оценивает трудозатраты при тестировании). Просим предоставить информацию, если недостаточно.
- Считать «тестов в день» для подсчета длительных задач (обычный рабочий день, когда приходится решать множество задач).

- Считать «минут та тест» и добавлять некоторый процент времени на решение остальных задач (чтение почты, вопросы коллег и прочие) для малых задач.
- Можно предоставить raw estimations, но обязательно добавить, что после получения ответов и/или определенных данных эстимации могут быть изменены.
- Учиться оценивать проще всего на практике, поэтому нужно эстимировать свои задачи сейчас.
- Стоит различать понятия **Estimations** и **Due Date**. Закладывать риски как на временные затраты, так и на сроки, особенно во время работы в команде.
- Не стоит бояться оглашать заказчику или начальнику большие трудозатраты, если они отражают действительность и можно доказать, обосновать необходимость каждого часа.
- Если не укладываетесь в определенные трудозатраты или в срок, то предупреждайте об этом заранее.
- Если не укладываетесь сделать задачу в отведенное время, анализируйте, что стало помехой.

Вопросы для обсуждения и задания:

1. Зачем нужно предоставлять эстимацию?
2. Как происходит эстимирование в Agile?
3. Если эстимированное время было 10 часов, риски, заложенные в эстимации, не сработали. Тестирующий потратил 10 часов на работу. Какова его производительность?
4. Какие методы эстимирования вы применяете?
5. Была ли у Вас ошибка во время подсчета и предоставления трудозатрат и как ее исправляли?

ТЕМА 3.2 НЕПРЕРЫВНАЯ ИНТЕГРАЦИЯ, ОСНОВЫ КОМАНДНОЙ СТРОКИ

Непрерывная интеграция (CI, Continuous Integration) – практика разработки программного обеспечения, которая заключается в постоянном слиянии рабочих копий в общую основную ветвь разработки (до нескольких раз в день) и выполнении частых автоматизированных сборок проекта для скорейшего выявления потенциальных дефектов и решения интеграционных проблем.

Выделяют следующие преимущества CI:

- Раннее обнаружение дефектов.
- Автоматическая сборка приложения. Как результат – приложение готово для тестирования.
- Мониторинг состояния продукта.
- Параллелизация.

Для непрерывной интеграции необходимы:

- Система версионного контроля.
- Автоматическая сборка билда.
- Автоматические тесты.
- Частые комиты.
- Сборка по каждому изменению.

Система управления версиями (Version Control System, Revision Control System, Source Code Management) – ПО, которое позволяет хранить и редактировать разные версии одного и того же документа, находить кто и когда сделал изменение. Примеры: Subversion, Git, Mercurial, Perforce.

Репозиторий (Repository, repo) – место (чаще всего доступное по сети), где хранятся какие-либо данные (чаще всего в виде файлов).

Ветвь (branch) – копия части или всего репозитория, в которую можно вносить изменения, не влияющие на другие ветви.

Фиксация изменений (check-in, commit) – сохранение изменений в ветви с присвоением этим изменениям уникального ID. Check-out – получение документа из хранилища и создание рабочей локальной копии.

Слияние (merge, integration) – объединение изменений из разных ветвей (возможно разными людьми) в единую версию документа.

Автоматизация сборки – этап процесса разработки программного обеспечения, заключающийся в автоматизации широкого спектра задач, решаемых программистами в их повседневной деятельности.

Сборщик (Builder) – это программа, позволяющая автоматизировать процесс превращения разрабатываемой программы из исходного кода в исполняемые файлы. Примеры сборщиков: Apache Ant, Apache Maven, Gradle.

Сервер непрерывной интеграции (CI Server) – программа, которая позволяет автоматизировать части и стадии процесса непрерывной интеграции. Примеры: Jenkins, Atlassian Bamboo.

Jenkins – программная система с открытым исходным кодом, предназначенная для обеспечения процесса непрерывной интеграции программного обеспечения.

Jenkins:

- Бесплатный.
- Кросс-платформенный.
- Имеет огромное количество плагинов (1300+).
- Очень широко распространен.

Непрерывная доставка (Continuous delivery, CD) – это подход к разработке программного обеспечения, при котором программное обеспечение производится короткими итерациями, гарантируя, что ПО является стабильным и может быть передано в эксплуатацию в любое время, а передача его не происходит вручную.

Целью является сборка, тестирование и релиз программного обеспечения с большей скоростью и частотой.

Консоль, интерфейс командной строки (Console, Command line interface, CLI) – разновидность текстового интерфейса между человеком и компьютером, в котором инструкции компьютеру даются в основном путём ввода с клавиатуры текстовых строк (команд).

Недостатки:

- Сложность для начинающих.
- Необходимость изучать/помнить команды.
- Непривычное отображение некоторых данных.
- Неудобство ввода длинных данных.

Зачем консоль тестировщику:

- Автоматизация рутинных действий.
- Ускорение работы.

- Доступ к «внутренностям» операционной системы.
- Более глубокое исследование тестируемых приложений и условий их работы.

Некоторые полезные команды cmd Windows:

Команда	Описание	Пример
cd	Сменить текущий каталог.	cd c:/123/
copy	Копировать файл(ы)	copy c:/123/*.txt d:/*.*
del	Удалить файл(ы)	del 1.txt
dir	Показать содержимое каталога.	dir
md	Создать каталог.	md NewDir
rd	Удалить каталог (пустой!)	rd NewDir
echo	Отобразить текст на экран.	echo мама мыла раму
set	Создать переменную.	set a=99
ipconfig	Найти свой ip адрес.	ipconfig
ping	Проверить интернет соединение с хостом.	ping tut.by
tracert	Отследить путь интернет запроса.	tracert tut.by
shutdown	Выключить/перезагрузить компьютер.	shutdown /s
netstat	Сетевые подключения.	netstat -a -o -n
Ctrl-C	Прервать операцию.	
taskkill	Завершить процесс.	taskkill /IM "calc.exe"

Некоторые полезные команды terminal Linux:

Команда	Описание	Пример
cd	Сменить текущий каталог.	cd /home
cat	Просмотреть файл(ы)	cat filename.txt
rm	Удалить файл(ы)	rm -i filename.txt
ls	Показать содержимое каталога.	ls -lh
cp	Скопировать файл в файл	cp -i file1 file2
chmod	Изменить доступ к файлу	chmod 777 file.txt
touch	Создать пустой файл.	touch myfile.ezh
mkdir	Создать каталог.	mkdir ~/temp
tar	Создать tar архив.	tar -czf archive.tar.gz folder
grep	Ищет в файле заданные строки.	grep -rl "string" /path
find	Найти файл по имени.	find /folder -name "*.txt"
ssh	Подключиться к удаленному компьютеру.	ssh tester@178.124.206.52
vim	Текстовый редактор.	vi file.txt
top	Информация о процессах (интерактивно).	top
ps	Информация о процессах (статическая).	ps aux

Варианты указания размещения файлов в terminal Linux:

Обозначение	Описание
.	Текущая папка
..	Родительская
~	Домашняя
~/	Файл в домашней
./	Файл в текущей

Пакетный файл (batch file) – текстовый файл в Windows, хранящий набор команд, которые выполняются интерпретатором командной строки.

Пример простого bat-файла:

```
@echo off
echo Hello! This is an example...
pause
md 123
dir
pause
rd 123
dir
pause
```

Особенности работы bat-файлов:

- Команды выполняются последовательно.
- Следующая команда выполняется после завершения предыдущей.
- Можно передать управление другому cmd или bat-файлу.
- Можно перенаправить вывод результата в файл.
- Можно получать параметры работы при запуске.

Цикл **FOR** позволяет работать с набором файлов:

FOR %переменная IN (набор) DO команда [параметры]

Составляющие:

%переменная – однобуквенный параметр.

(набор) – набор, состоящий из элементов, обрабатываемых командой.
команда – то, что следует выполнить для каждого элемента набора.
параметры – параметры для команды
%%переменная вместо %переменная – в .bat файлах

В цикле **FOR** можно оперировать названиями и атрибутами файлов:

%~n – убрать окружающие кавычки	%~fn – получить полное имя
%~dn – получить букву диска	%~pn – получить путь
%~nn – получить имя файла	%~xn – получить расширение
%~an – получить атрибуты	%~tn – получить дату/время создания
%~zn – получить размер	

В bat-файлах можно создавать условные конструкции:

If УСЛОВИЕ (действие1) [else (действие2)]

Вопросы для обсуждения и задания:

1. Дайте определение непрерывной интеграции и непрерывной доставки
 2. Какие преимущества дает проекту непрерывная интеграция?
 3. Что такое bat-файл?
 4. Для чего может понадобиться bat-файл ручному тестировщику?
 5. Как можно подключиться к удаленному компьютеру в консоли?
-
-
-
-
-

IT-Academy

ТЕМА 3.3 ПОДГОТОВКА РАБОЧЕГО МЕСТА WINDOWS

Виртуальная машина (virtual machine) – система, эмулирующая аппаратное обеспечение некоторой платформы и исполняющая программы для этой платформы.

Гостевая платформа (guest) – исполняется под виртуальной машиной.

«Хостовая» (основная) платформа (host) платформа – исполняет виртуальную машину.

Применение ВМ в тестировании:

- исследование производительности ПО;
- быстрое создание тестовой среды с заданными параметрами;
- проверка потенциально опасных ситуаций;
- тестирование совместимости (кроссплатформенное, кроссбраузерное).

Данные виртуальной машины можно хранить в одном файле. Так перенастраивая одну и ту же виртуальную машину можно получить несколько машин с разными настройками ОС (языки, драйверы и т.п.), разное программное окружение, разные версии тестируемого продукта и любые другие эксперименты. Так же «образ» с найденной проблемой можно целиком отдать разработчикам.

Гипервизор (Hypervisor; от др.-греч. ὑπέρ «над, выше, сверх» + лат. vīsiō «зрение; видение») или монитор виртуальных машин (в компьютерах) – программа или аппаратура, позволяющая одновременный запуск нескольких операционных систем на одном и том же хост-компьютере.

Для создания виртуальной машины:

- Нужно установить на компьютер гипервизор (VirtualBox, VMWare Player).
- После этого в нем создать саму виртуальную машину.
- На виртуальную машину установить желаемую операционную систему.

Services (службы) – приложения, автоматически (если настроено) запускаемые системой при запуске Windows, выполняющиеся в фоне и как правило не имеющие GUI. Аналогичные программы в Unix называют демонами.

Где посмотреть

Control Panel -> All Administrative Tools -> Services

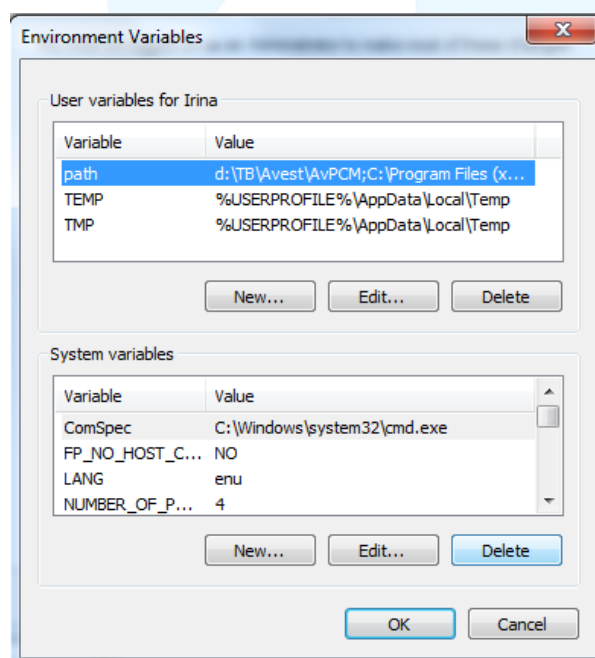
Если выбрать службу, то через контекстное меню есть возможность открыть общие свойства службы, где можно узнать путь к исполняемому файлу службы, изменить режим запуска службы, изменить текущее состояние службы. Вкладка Dependencies позволяет исследовать взаимные зависимости между анализируемой службой и другими службами в системе.

Переменные окружения (environment variables) – текстовые переменные операционной системы, хранящие данные о ряде настроек системы.

Где посмотреть

My Computer -> в контекстном меню Properties -> Advanced system settings -> Environment Variables.

Control Panel -> All Control Panel Items – > System -> Advanced system settings -> Environment Variables.



Task Manager (Диспетчер задач) – встроенная в операционную систему Windows утилита для вывода на экран списка запущенных процессов и потребляемых ими ресурсов (например, статус, процессорное время и потребляемая оперативная память). Также есть возможность некоторой манипуляции процессами.

Имеет следующие вкладки:

- Приложения. Позволяет переключиться в нужное приложение, либо завершить его. Можно воспользоваться этим окном при необходимости, если какая-либо из программ «не отвечает», и закрыть её принудительно ("Снять задачу").
- Процессы. Различные данные обо всех запущенных в системе процессах, где можно завершать, менять приоритет, задавать соответствие процессорам (в многопроцессорных системах). Можно воспользоваться опцией "Завершить процесс" при "подвисании" какого-либо процесса.
- Службы. Сведения обо всех службах Windows.
- Быстродействие. Графики загрузки процессора (процессоров), использования оперативной памяти.
- Сеть (отсутствует, если нет активных сетевых подключений). Показывает графики загрузки сетевых подключений.
- Пользователи (только в режиме администратора). Манипулирование активными пользователями.

Где посмотреть

Ctrl+Alt+Del -> выбираем Task Manager (Диспетчер задач).

Resource Monitor (монитор ресурсов) – встроенная в операционную систему Windows утилита для наблюдения и манипуляции процессов и потребляемых ими ресурсов (например, статус, процессорное время и потребляемая оперативная память).

Event Log (журнал событий) – в Microsoft Windows стандартный способ для приложений и операционной системы записи и централизованного хранения информации о важных программных и аппаратных событиях. Программа просмотра событий позволяет пользователю наблюдать за журналом событий.

Process Monitor – бесплатный инструмент для операционных систем Microsoft Windows, который предоставляет пользователям возможность для мониторинга файловой системы, системного реестра, а также всех процессов в оперативной памяти в реальном времени.

Реестр Windows, системный реестр (registry) – иерархически построенная база данных параметров и настроек в ОС Windows. Реестр содержит информацию и настройки для аппаратного обеспечения, программного обеспечения, профилей пользователей. Большинство изменений в панели управления, ассоциации файлов, системные политики, список установленного ПО фиксируются в реестре.

Удаленный рабочий стол через Remote Desktop Protocol

Remote Desktop Protocol – это протокол прикладного уровня, который обеспечивает доступ к удаленным компьютерам без непосредственного контакта с ними. Пользователь сможет видеть файлы, запускать программы и выполнять все различные задачи так, как будто он находится в непосредственной близости от компьютера.

В операционной системе Windows существует программное средство для настройки удаленного рабочего стола (Remote Desktop Connection), которое работает по описанному выше протоколу и обеспечивает следующие функции:

- Поддержку полноценной передачи цвета.
 - Возможность использовать современные средства шифрования.
 - Возможность подключения локальных ресурсов к удаленному компьютеру. (Например, можно подключить жесткий диск для обмена данными между локальным компьютером и удаленным).
 - Возможность использования на удаленном устройстве локального и сетевого принтера.
 - Переадресацию и воспроизведение звука с удаленного устройства на локальном компьютере.
 - Возможность обмениваться данными через буфер обмена (например, скопировать текст на одном устройстве и вставить в документ на другом).
-
-

Подключение:

1. "Пуск" -> ввести «mstsc» в строку поиска и нажать Enter.
2. Ввести IP адрес в поле "Компьютер" в окне "Подключение к удаленному рабочему столу" -> "Подключить".

Можно воспользоваться другими достаточно популярными инструментами для организации удаленного подключения:

TeamViewer, WebEX, LogMeIn, Google Chrome Remote Desktop, LapLink, RealVNC, GoToMyPC.

Полезные инструменты и приложения

Problem Steps Recorder – стандартное приложение Windows, которое позволяет записать шаги пользователя в виде изображений. Ввести PSR в поле поиска в Windows и нажать Enter.

Fiddler – бесплатный прокси сервер, который работает с трафиком между вашим компьютером и удаленным сервером, и позволяет просматривать и менять его.

Вопросы для обсуждения и задания:

- Как можно провести кроссплатформенное тестирование, не имея нескольких компьютеров?
- Как можно узнать почему у вас «тормозит» компьютер?
- Расскажите, каким образом можно подключиться к удаленному компьютеру?
- Какие программы, Web-приложения используете во время выполнения работы за компьютером, оптимизирующие ваши действия?

IT-Academy

ТЕМА 3.4 МЕТОДОЛОГИИ РАЗРАБОТКИ ПО

Методология разработки ПО (Software Development Process, Software Development Life Cycle) – схема/стандарт/правила определяющая характер и последовательность выполнения процессов, действий и задач по разработке программного средства на протяжении периода времени (жизненного цикла), который начинается с момента принятия решения о необходимости создания программы и заканчивается в момент прекращения ее использования.

Каскадная модель (waterfall model, иногда переводят как модель «Водопад») – методология разработки ПО, в которой процесс разработки выглядит как поток, последовательно проходящий все фазы.

В каскадной модели фазы идут в следующем порядке:

- Определение требований.
- Проектирование.
- Конструирование (также «реализация» либо «кодирование»).
- Воплощение.
- Тестирование и отладка (также «верификация»).
- Инсталляция.
- Поддержка.

Переход от одной фазы к другой происходит строго последовательно и только после полного и успешного завершения предыдущей.

Итеративный подход (iteration – «повторение») в разработке ПО – это подход, при котором выполнение работ происходит циклично (цикл PDCA, plan-do-check-act: Планирование – Реализация – Проверка – Оценка) и параллельно с постоянным анализом полученных результатов и корректировкой предыдущих этапов.

Преимущества итеративного подхода:

- Снижение проектных рисков, что минимизирует затраты на их устранение.
 - Наличие обратной связи от потребителей или заказчиков, что позволяет создать продукт, реально отвечающий их потребностям.
 - Непрерывное повторяющееся тестирование и как следствие лучшее качество.
 - Более равномерная загрузка команды проекта.
-
-

Гибкая методология разработки (Agile software development) – совокупность методов и подходов к разработке ПО, которые основаны на итеративности, динамическом создании требований и их реализации в процессе постоянного взаимодействия

небольшой команды с заказчиком. К таким методикам можно отнести XP, DSDM, Scrum, Kanban, BDD.

SCRUM (scrum «схватка») – метод управления проектами в сфере разработки ПО, относящийся к семейству гибких методологий.

Роли в SCRUM:

- **Скрам-мастер** (SCRUM Master) – специалист, задача которого нивелировать проблемы и противоречия в команде, минимизировать отвлекающие моменты, следить за соблюдением правил скрама и проводить совещания (Scrum meetings).
 - **Скрам-владелец продукта** (Product Owner) – специалист, который представляет интересы заказчика и/или конечных пользователей.
 - **Скрам-команда** (Team) – команда, состоящая из 5 – 9 человек специалистов разных профилей (разработчики, тестировщики, BA и др.).
-
-
-
-

Артефакты SCRUM:

- **Журнал пожеланий проекта** (Project backlog) – это набор требований (user story) к ПО, отсортированный по важности. За ведение Project backlog отвечает Product Owner.
 - **Журнал пожеланий спринта** (Sprint backlog) – это выборка требований из журнала пожеланий проекта, которую команда должна реализовать в рамках текущего спринта.
 - **История пользователя** (User Story) – это формат написания требований, представляющий собой одно предложение вида «Как <пользователь> я хочу сделать <действие>, чтобы получить <ценный результат>».
 - **Очки за пользовательскую историю** (Story Points) – это абстрактные баллы, которые показывают относительную сложность задачи. Часто используют такие шкалы, как ряд Фибоначчи (1,2,3,5,8,13,21), степень двойки (1,2,4,8 ... 2n), размеры одежды (XS, S, M, L, XL).
 - **Диаграмма сгорания задач** (Burndown chart) – график, который показывает динамику уменьшения оставшейся работы во времени.
-
-
-
-

Собрания SCRUM:

- **Планирование спринта** (Sprint Planning Meeting) – собрание, предназначенное для выбора из бэклога проекта задач, обязательных к выполнению за грядущий

спринт. **Покер планирования** (Planning Poker) – техника оценки сложности предстоящей работы разработке ПО, проводимая в Story Points.

- **Ежедневное SCRUM-совещание** (Daily SCRUM) – ежедневная короткая (на 15 минут) встреча SCRUM Master, Product Owner Team, где каждый отвечает на три вопроса:
 - Что я сделал вчера?
 - Что я сделаю сегодня?
 - Что мне мешает что-то сделать?
- **Обзор итогов спринта** (Sprint review meeting) – совещание, которое проводится в конце спринта, где команда показывает заинтересованным лицам новую часть приложения (Demo) и получает обратную связь.
- **Причесывание беклога** (Backlog grooming meeting) – совещание, на котором команда может добавить, убрать User Story, разбить существующие на более мелкие, уточнить их оценки и/или переприоритезировать.
- **Ретроспективное совещание** (Retrospective meeting) – совещание, на котором всеми высказывается мнение о прошедшем спринте, чтобы в последующем улучшить процесс разработки или избежать известных проблем.

Вопросы для обсуждения и задания:

1. Назовите стадии каскадной модели.
2. Назовите преимущества и недостатки каскадной модели.
3. Какие преимущества есть у итеративного подхода?
4. Назовите артефакты Scrum.
5. Какую методологию разработки Вы бы выбрали для нового проекта?

ТЕМА 3.5 ОСОБЕННОСТИ МОБИЛЬНОГО ТЕСТИРОВАНИЯ

Mobile application (мобильное приложение) – программное обеспечение, предназначенное для работы на смартфонах, планшетах и других мобильных устройствах. Многие мобильные приложения предустановлены на самом устройстве или могут быть загружены на него из онлайн-магазинов приложений, таких как Play Market, App Store, Windows Market.

Мобильная операционная система (мобильная ОС) — операционная система для смартфонов, планшетов, КПК или других мобильных устройств. Хотя ноутбуки и можно отнести к мобильным устройствам, однако операционные системы, обычно используемые на них, мобильными не считаются, так как изначально разрабатывались для крупных стационарных настольных компьютеров, которые традиционно не нуждались в специальных «мобильных» функциях. Это различие размыто в некоторых новых операционных системах, представляющих гибрид того и другого.

Современные операционные системы для мобильных устройств: Android, Kai OS, Lineage OS, Fire OS, Flyme OS, iOS, Sailfish OS, Tizen, Remix OS.

Устаревшие, ныне не поддерживаемые программные платформы: Windows 10 Mobile, Symbian, Windows Mobile, Palm OS, webOS, Maemo, MeeGo, LiMo, BlackBerry OS, Firefox OS, Ubuntu Touch, Bada OS.

Виды мобильных приложений

Native app – приложение, которое разрабатывается с помощью технологий, относящихся к платформе, для которой разрабатывается приложение (например, Android, iOS, Windows). Так, например, приложения для iOS могут разрабатываться с использованием Object C, Swift, а для Android с использованием Java. Нативные приложения могут использовать функционал GPS, видеокамер или датчиков ускорения. Главным преимуществом нативных приложений является возможность автономной работы без необходимости подключения к интернету.

Mobile Web app – приложение, использующее технологию WEB для работы на мобильном устройстве. Главным преимуществом таких приложений является возможность единовременного создания на все типы платформ с помощью HTML5 (кроссплатформенность). Такие приложения невозможно загрузить из магазина приложений, они доступны в браузере мобильного устройства и не позволяют использовать функции камеры или геолокации в смартфоне.

Hybrid app – приложение, использующее такие функции нативных и веб-приложений, как кроссплатформенность и возможность использования ПО телефона (доступ к камере, GPS). Гибридные приложения загружаются через магазин приложений, но при этом в них происходит обновление Веб-контента через Интернет. Пользователи не замечают особенную разницу между нативным и гибридным приложениями. Всем известные мобильные приложения социальных сетей разработаны как гибридные приложения.

Адаптивный веб-дизайн (Adaptive Web Design, Responsive Web Design) – технология создания веб-страниц, обеспечивающая правильное отображение страниц сайта на устройствах с экранами различных разрешений (смартфоне, планшете, ноутбуке и т.п.) путем динамического подстраивания под размеры окна браузера.

Прогрессивное веб-приложение (progressive web app, PWA) – технология в веб-разработке, которая добавляет сайтам возможности мобильных приложений, такие как установка на мобильное устройство, офлайн режим, наличие пуш-уведомлений.

Android Instant App – легковесная и ограниченная версия мобильного приложения, позволяющая пользователю быстро увидеть примерный функционал приложения.

Ключевые моменты в мобильном тестировании:

- Понять основную функциональность будущего ПО.
 - Проанализировать статистику используемых платформ.
 - Выбрать оптимальный набор аппаратов + операционных систем.
 - Решить где взять устройства.
 - Учесть специфические тесты для мобильных устройств.
-
-

Возможными вариантами использования разнообразных платформ могут быть физические устройства, удаленный доступ, эмуляторы и симуляторы.

Эмулятор (Emulator) – это комплекс аппаратных средств и/или программ, который позволяет скопировать (эмулировать) функционал одной системы (гостя) на другой

(хосте), максимально точно соответствуя оригиналу. Примеры: виртуальные машины, эмулятор Android.

Симулятор (Simulator) – это комплекс аппаратных средств и/или программ имитирующий процесс, программу или аппарат, но таковым не являющийся. Примеры: игровые симуляторы, симулятор Xcode.

Специфичные тесты для мобильных приложений

Прерывания работы приложения:

- Реакция на звонки, SMS, MMS, оповещения, уведомления других приложений.
 - Реакция на выключение, разрядку, подключение к сети, зарядка.
 - Переход в режим ожидания.
 - Смена ориентации устройства в режиме ожидания.
-
-

Установка и обновления:

- Установка, удаление, переустановка, обновление.
 - Проверка сохранения настроек и данных приложения после обновления.
 - Проверка установки обновлений при различных соединениях.
 - Регрессия старой функциональности в новой версии.
 - Попытка установки обновления при недостатке места для обновления.
-
-

Особенности работы с сетью:

- Различные виды сети (2G, 3G, LTE, WIFI).
 - Различное качество связи.
 - Переходы с одной сети в другую.
-
-

Отклик в приложении и производительность:

- Проверка корректного отображение нажатого состояния элементов.
 - Проверка скорости отклика кнопок при нажатии.
 - Проверка синхронизации звуков, видео, вибраций.
 - Проверка использования CPU, оперативной памяти, диска, батареи.
 - Проверка работы при установке на внешнюю карту памяти.
-
-

Операционные возможности устройств

- Одновременная работа нескольких приложений.
- Проверка работы приложения при отсутствии определенных элементов (3G, SD-карта и другие).
- Проверка утечки памяти (кэширование).
- Проверка нативных жестов, предусмотренных функционалом.
- Работа с геолокацией, камерой, ориентацией, акселерометром и др.

Вопросы для обсуждения и задания:

1. Какие знаете платформы мобильных устройств?
2. Как установить приложение на мобильное устройство?
3. Какие специфичные тесты необходимо проводить при тестировании мобильных устройств?
4. Каким образом можно протестировать приложение на мобильном устройстве, если конкретного мобильного устройства нет в наличии?

IT-Academy

ТЕМА 3.6 ОСОБЕННОСТИ ТЕСТИРОВАНИЯ ЛОКАЛИЗАЦИИ, ОСНОВЫ ТЕСТИРОВАНИЯ ДОСТУПНОСТИ

Интернационализация (i18n, internationalization) – использование различных технологических приёмов во время разработки приложения, которые позволяют быстро адаптировать приложение под различные языковые и культурные особенности региона. В английском языке для слова «internationalization» принято сокращение «i18n»

Локализация (l10n, localization) – процесс адаптации программного обеспечения к культуре и языку какой-либо страны. Как частность – перевод пользовательского интерфейса, документации и сопутствующих файлов программного обеспечения с одного языка на другой. Для локализации в английском языке применяют сокращение «L10n».

Псевдо – локализация (pseudo-translation) – это один из методов проверки готовности приложений к локализации, то есть проверка того, что в приложении выполнена интернационализация и когда приложение переведут для какого-либо региона (выполнят локализацию), то все необходимые элементы будут переведены и будут выглядеть приемлемо.

Региональный стандарт, региональные настройки (проф. жарг. локаль) – набор параметров, определяющий региональные настройки пользовательского интерфейса, такие как язык, страна, часовой пояс, набор символов и т. п.

Что подвергается локализации:

- Текст программ (названия полей, кнопок, подсказки, сообщения об ошибках)
- Графика и мультимедиа
- Комбинации клавиш
- Шрифты
- Документация
- Сопутствующие материалы
- Шаблоны документов
- Иллюстрации.

Варианты тестов локализации

Формат даты	English: mm/dd/yy Spanish: dd/mm/yy Japanese: yy/mm/dd English: Tuesday, October 12, 1954 Spanish: martes 12 de octubre de 1954 Japanese: 1954年10月12日
Формат времени	US English – 10:27:56 PM UK English – 22:27:56
Валютное обозначение	Один символ '€' (Euro) Несколько символов 'GBP' (British Pound) Расположение знака валюты до суммы или после нее.
Отрицательные значения	UK -£127.54 France -127,54 F Denmark kr-127,54 US (\$127.54) Netherlands € 127,54-
Формат чисел	Разделитель тысяч, десятичной части и группировка. USA – (,) 1,025 Germany – (.) 1.025 Russian – 1 025 USA – 123,456,789.00 Hindi – 12,34,56,789.00 USA – (.) 1,025.7 Germany – (,) 1.025,7
Формат адреса	Индекс/Страна/Город/Улица Улица/Город/Страна/Индекс
Увеличение длины строки и высоты	В некоторых языках (финский, немецкий и голландский) вместо последовательности коротких слов создается длинное слово «Input processing features» (Функции обработки ввода) «Eingabeverarbeitungsfunktionen» (на немецком языке)
Аббревиатуры	В некоторых языках могут не использовать аббревиатуру или появятся ошибки во время перевода.
Порядок слов	Разный порядок слов: Имя Фамилия Фамилия Имя

Accessibility (A11y, «доступность», «общедоступность») выявляет степень доступности продукта, устройства или сервиса для разных пользователей, включая пользователей с различными органическими или техническими ограничениями.

Тестирование доступности (Accessibility Testing) – тестирование, направленное на исследование пригодности продукта к использованию людьми с ограниченными возможностями.





Тестирование доступности:

Масштабирование (Zoom)	
Цель проверки: <ul style="list-style-type: none">• Размер шрифта текста и размер иконок может быть увеличен до 200%.• Появление скролл бара.• Расположение и порядок чтения остается неизменным.	Инструмент: функция Zoom в браузере.
Навигация при помощи клавиатуры	
Цель проверки: <ul style="list-style-type: none">• Возможность работы при помощи клавиатуры.• Соблюден порядок работы: слева направо, сверху вниз.	Основные клавиши: <ul style="list-style-type: none">• Tab – следующий элемент.• Shift+Tab – предыдущий элемент.• Enter или Space – выбрать элемент, чек-бокс.• Arrows up, down, left, right.
Evaluation tool	
Цель проверки: Обеспечить визуальную обратную связь о доступности вашего веб-контента, вставляя значки и индикаторы на страницу.	Инструмент: WAVE Evaluation Tool/ tota11y/ Siteimprove/ axe/ Lighthouse
Screen Readers	
Цель проверки: <ul style="list-style-type: none">• Пользователь может передвигаться по элементам.• При передвижении по элементам screen reader зачитывает все активные элементы на странице.• Заголовки на странице озвучиваются для упрощения понимания структуры страницы.• Тип и состояние элемента называются корректно (Radio buttons/Check boxes, drop-downs, etc.).• Alerts/Popup messages сообщаются пользователю.	Инструмент: JAWS, NVDA, ChromeVox
Color Contrast Ratio	
Цель проверки: <ul style="list-style-type: none">• Контраст цвета между бэкграундом и текстом достаточен для восприятия людьми с ослабленным зрением.• Контраст должен быть как минимум: 4.5:1 для короткого текста и 3:1 для длинного текста.• Информация не должна передаваться только цветом.	Инструмент: Color Contrast Analyzer

Вопросы для обсуждения и задания:

1. Каким образом реализуется возможность адаптации приложения на другой язык?
2. Что необходимо тестировать во время локализации приложения?
3. Приведите примеры инструментов, которые можно использовать для тестирования доступности веб приложения
4. Чем отличаются уровни соответствия стандарту доступности А, АА и ААА?

IT-Academy

 +375 (29) 222-24-60 +375 (44) 570-22-22 +375 (25) 760-24-60 info@it-academy.by Минск, ул. Скрыганова, 14,
5-й этаж, каб. 59