ЛАБОРАТОРНАЯ РАБОТА №1

Тема лабораторной работы: тестирование документации.

Общие сведения по работе

Документация является одним из объектов тестирования. Главной целью в этом тестировании является сокращение ошибок разного характера, улучшение структурированности, предупреждение потенциальных проблем в дальнейшей разработке программного обеспечения. Настоящая лабораторная работа направлена не столько на исправление документации для ее дальнейшего использования, сколько для проведения ретроспективного анализа своей собственной работы, результат которой уже известен, и модификации документации для повышения ее качества. Необходимо протестировать документацию по одной из своих прошлых работ (курсовая работа или проект).

Методические рекомендации и материалы

Документация является одним из составных компонентов любой программной системы. Документация в зависимости от вида жизненного цикла (ЖЦ) программного обеспечения может создаваться на каждом из этапов. Контроль за качеством документации является одной из актуальных проблем в разработке.

Тестирование документации проводится для:

1. Минимизации потенциальных рисков при реализации новой функциональности.
2. Минимизации проблем доработки и сопровождения существующего программного обеспечения.

Определение доступной для тестирования документации можно выполнить, основываясь на наличии формальных требованиях к ней. Можно выделить две группы документации:

1. Документация, выполненная в строгом соответствии со стандартами (ГОСТ, ISO): техническим заданием (ТЗ), спецификациями, требованиями к программному обеспечению, инструкциями, и т. д.
2. Документация, выполненная по внутренним правилам команды или заказчика (документация, основанная на стандартах, сгенерированная документация).

Тестировать документацию можно, основываясь на следующих критериях:

1. Полнота. Каждый элемент функциональности должен был представлен в документе в требуемом объеме. Если описываются функциональные требования в ТЗ, то они должны давать полное представление, как должна работать система.
2. Однозначность. Интерпретация написанного в документе должна быть одинаковой для всех участников проекта. Необходимо приводить списки терминов и аббревиатур с расшифровкой.
3. Непротиворечивость. Документ проверяется на наличие конфликтных требований, на соответствие требований в разных разделах документа.
4. Актуальность. Документация и реально разработанный программный продукт соответствуют друг другу в один и тот же момент времени.
5. Структурированность. Структурные элементы документации позволяют пользователю осуществлять поиск нужной ему информации.
6. Тестируемость. Если описывается какая-либо функциональность, то должна быть возможность ее проверки на финальном этапе разработки. Если протестировать что-то невозможно, то каким образом будет проверяться завершенность разработки?

Использование документации не только для процесса разработки, но и процедуры тестирования важно с точки зрения следующих аспектов:

1. Для ускорения процедуры тестирования. От детальности и точности описания функциональности зависит то, как быстро тестировщик сможет выполнить тестирование.
2. Для использования полной документации в качестве источника для создания тестовой документации и проектирования тест-планов.
3. Для сокращения затрат на техническую поддержку.

Любая деятельность по тестированию направлена проверку объектов тестирования при помощи соответствующих методов, определяющих порядок операций с объектом. Для тестирования документации применяются следующие методы:

1. Рецензирование (анализ) документации разными группами специалистов. Анализ позволяет выявить проблемы, с которыми знаком каждый из экспертов своей области. Рецензирование выполняется путем просмотра пакета документов специалистом и фиксации выполнения заданных критериев тестирования документации.
2. Пробное создание тест-кейсов (сценариев тестирования). Это трудоемкий вариант, но с его помощью возможны модельные прогоны сценариев для выявления недостатков, устранения противоречий. Собираются сведения о необходимости корректировок, уточнений.
3. Обсуждение на основе проектов и прототипов. Создаются наброски (структурные, функциональные, UI, …), позволяющие детализировать критические моменты системы и позволяющие в схематичном графическом виде удобно представить некоторые части

будущей системы. Результатом обсуждения будет пересмотр части пунктов документации.

Порядок тестирования документации:

1. Определить объем и вид документов, требующихся для конкретного вида жизненного цикла разработки и выбранной методологии разработки.
2. Определить цели тестирования документации.
3. Выбрать существующие документы и провести их анализ одним из выбранных способов.
4. Оценить документацию по критериям, сформировать отчет о тестировании документации.

Задания к лабораторной работе

1. Выбрать документацию, в которой описываются требования к проектированию или разработке программного обеспечения или структуре базы данных.
2. Провести анализ и составить отчет со следующей структурой:
    a. Цель работы.
    b. Описание тестируемой документации.
    c. Описание критериев качества тестируемой документации
    d. Описание и обоснование метода тестирования документации.
    e. Список несоответствий в документации критериям качества с указанием номеров пунктов исходной документации, цитированием части документации и конкретными рекомендациями к исправлению.
    f. Выводы по работе.
    g. Список использованных источников.
3. Оформить и защитить отчет.

Контрольные вопросы

1. Объясните понятие «жизненный цикл программного обеспечения».
2. Какую документацию нужно тестировать?
3. На каком этапе ЖЦ разработки нужно тестировать документацию?
4. Из каких шагов состоит тестирование документации?
5. Когда тестирование документации оправдано?

ЛАБОРАТОРНАЯ РАБОТА №2

Тема лабораторной работы: работа с классификацией видов тестирования.

Общие сведения по работе

Классификация видов тестирования отражает многообразие походов организации контроля качества программных систем. Понимание классификации позволяет определить возможные варианты процедур тестирования, оценить трудоемкость, спланировать тестирование, выбрать подходящие инструменты. Это знание является необходимым при построении жизненного цикла разработки программных систем. Навыки использования видов тестирования для контроля функционирования программных систем позволяют тестировщику быстрее локализовать дефекты, создавать более полные наборы тестов.

Методические рекомендации и материалы

Деление на различные виды тестирования позволяет применять только подходы соразмерно требованиям к проверке поставленных требований, т. е. отдельные виды тестирования могут получить повышенный приоритет выполнения, в то время как все прочие виды тестирования будут не так важны. Выбор видов тестирования зависит от поставленных требований (зачастую в ТЗ уже сформулированы требования приемочного тестирования, которые нужно будет проверить при готовности системы), этапа разработки, условий использования системы. Используемые виды тестирования можно использовать в чистом виде (например, тестирование пользовательского интерфейса), так и охарактеризовать с позиций типов, методов и других видов тестирования (ручное функциональное тестирование, автоматизированное функциональное тестирование, регрессионное ручное функциональное тестирование, и т. д.)

Представление классификации основано на делении следующим ключевым группам:

1. Уровни тестирования предполагают оценивать программную систему с точки зрения объема программных модулей и учета соответствующих аспектов (например, внимание будет сосредоточено только на проверке кода, реализующего бизнес-логику, или важным будет являться момент интеграции подсистем или модулей). В зависимости от степени детальности компонентов подразделяются на:

– Компонентное (модульное) – тестирование программы на уровне отдельно взятых модулей, функций или классов. Цель компонентного тестирования – выявление локализованных в модуле ошибок в реализации алгоритмов, а также определение степени готовности системы к переходу на следующий уровень разработки и тестирования.

– Интеграционное тестирование предназначено для проверки связи между компонентами, а также взаимодействия с различными частями системы (операционной системой, оборудованием либо связи между различными системами). Уровнями интеграционного тестирования являются: компонентный интеграционный (проверяется взаимодействие между компонентами системы после проведения компонентного тестирования), системный интеграционный уровень (проверяется взаимодействие между разными системами после проведения системного тестирования).

– Системное тестирование проверяет как функциональные, так и нефункциональные требования к системе в целом. При этом выявляются дефекты, такие как неверное использование ресурсов системы, не предусмотренные комбинации данных пользовательского уровня, несовместимость с окружением, непредусмотренные сценарии

использования, отсутствующая или неверная функциональность, неудобство использования и т. д.

При этом деление на уровни тестирования предполагает участие различных ролей участников команды разработки. Предполагается, что тесты уровня компонентного тестирования разрабатывает сам программист, контролируя самостоятельно качество своей работы или работы своих коллег. Интеграция модулей или общесистемное тестирование могут быть проведены уже другими участниками команды, знающими требования к программным интерфейсам модулей и подсистемам и имеющими возможность для манипуляции с ними.

2. Типы тестирования сосредоточены на делении видов тестирования, проверяющих наличие требуемой функциональности (функциональный тип тестирования, что система делает) и проверяющих режимы работы системы при выполнении пользователями своих задач (нефункциональный тип тестирования, как система это делает).

Функциональный тип тестирования включает следующие подвиды:

– функциональное тестирование;

– тестирование возможности взаимодействия (включает тестирование совместимости и интеграционное тестирование);

– тестирование безопасности.

Нефункциональный тип тестирования описывает такие подвиды, как:

– нагрузочное тестирование;

– стресс-тестирование;

– тестирование удобства пользования;

– тестирование надежности и т. д.

Примеры проведения нефункциональных тестов:

– общее исследование поведения системы при экстремальных нагрузках;

– экспертиза обработки ошибок и исключений;

– тестирование пропускной способности системы;

– изучение некоторых частей системы или ее компонентов в условиях непропорциональной нагрузки.

3. Методы тестирования описывают применение видов тестирования в зависимости от объекта тестирования. Методы подразделяются на статические (тестирующие такие объекты, как программный код, документацию, ресурсы проекта, все то, что может быть проверено без непосредственного выполнения программного кода, т. е. запуска программной системы) и динамические (тестирующие функциональность программной системы и режимы ее эксплуатации).

Подвидом статического метода тестирования является рецензирование – вид тестирования, который может проводиться перед динамическим тестированием.

То, как он может быть проведен:

– вручную,

– с применением анализаторов кода.

Рецензирование может проводится для любого продукта, связанного с разработкой программного обеспечения, включая спецификации требований и дизайна, код, планы тестирования, руководства пользователя и т.п. Во время рецензирования могут быть найдены упущения, например в требованиях, которые маловероятно найти во время динамического тестирования. Более широким видом статического тестирования является статический анализ, проводимый при помощи специальных инструментов. При этом анализируется:

– код программы (например, потоки управления и поток данных),

– сгенерированный код, например HTML, XML.

На рисунке 1 приведен пример скриншота отчета статического анализатора кода SonarQube с результатами анализа, охарактеризованных по степени серьезности, с рекомендацией к устранению. На рисунке 2 проводится отчет о валидации html разметки сайта ulstu.ru.

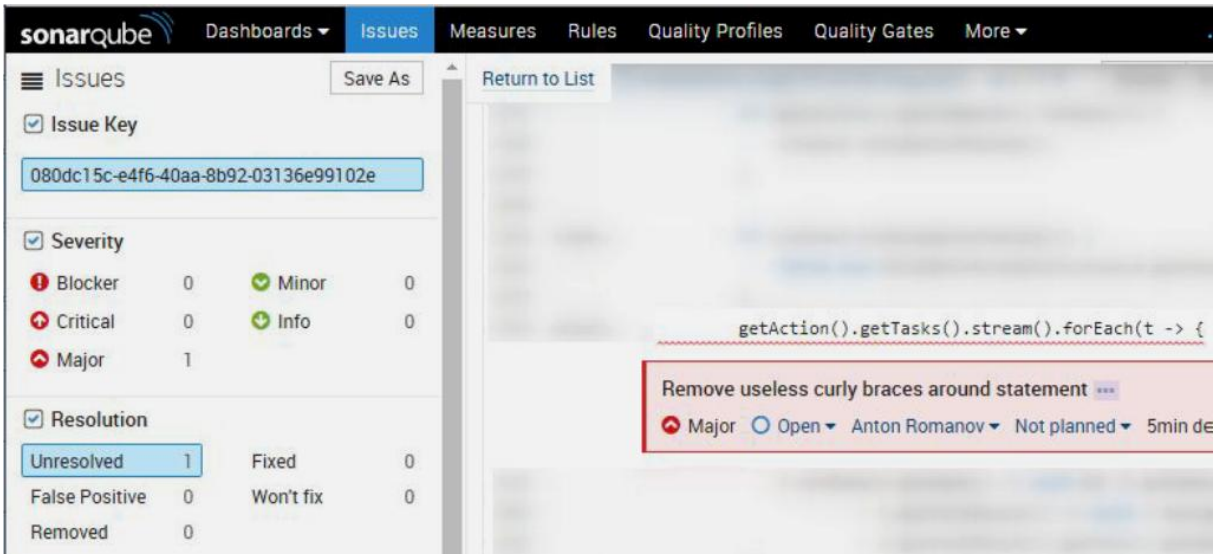

Рис. 1. Отчет в системе SonarQube о статическом анализе программного кода проекта

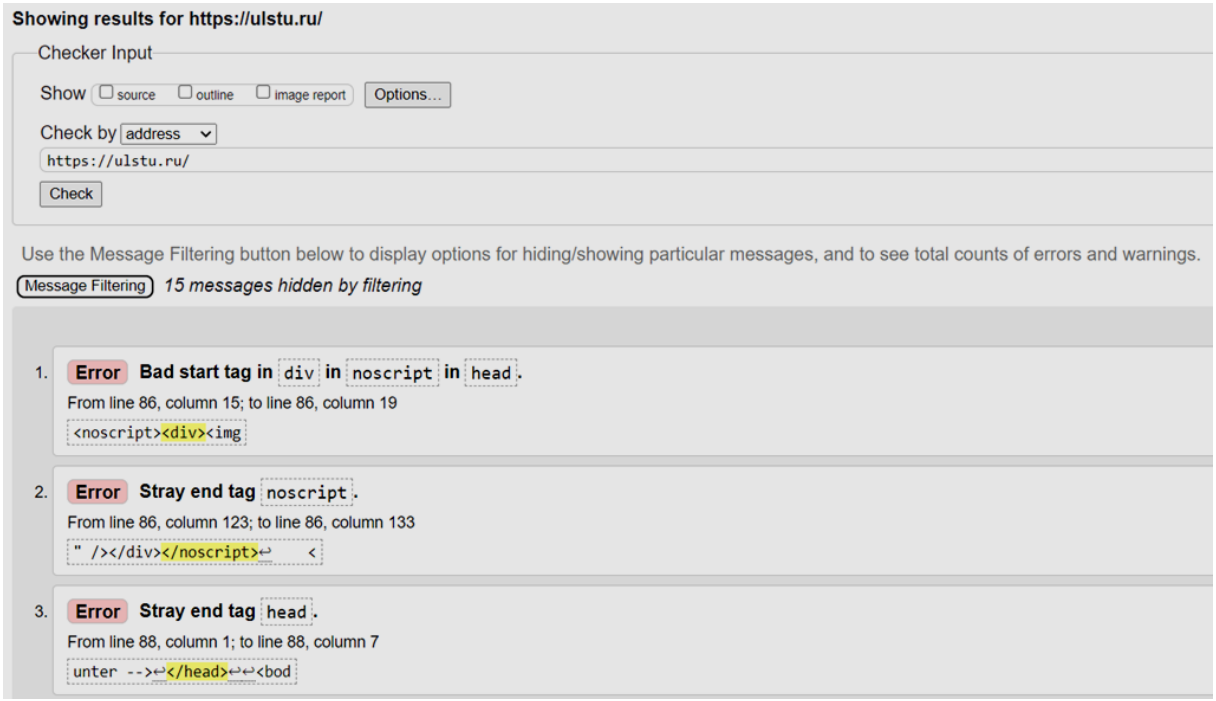

Рис. 2. Отчет валидации html разметки сайта

Динамические методы тестирования позволяют проверять программную систему при ее непосредственном выполнении. Динамические методы делятся на:

– Методы белого ящика. Тестирование на соответствие программного продукта требованиям со знанием внутренней структуры реализации системы (есть в наличии исходный код и технические спецификации). Это вид тестирования позволяет проводить локализацию ошибок, анализ надежности и устойчивости и другие типы проверок, существенно повышая качество системы.

– Методы черного ящика. Тестирование на соответствие программного продукта требованиям без знания внутренней структуры реализации системы.

– Методы серого ящика. Комбинирование методов белого и черного ящика. При тестировании серого ящика разработчик теста имеет доступ к исходному коду, но при непосредственном выполнении тестов доступ к коду, как правило, не требуется.

4. Виды тестирования, подразделяются на группы:

- по объекту (на что обращен процесс тестирования).
    - Функциональное. Проверяется соответствие требованиям. Оценка производится в соответствии с ожидаемыми и полученными результатами (на основании функциональной спецификации), при условии, что функции отрабатывали на различных значениях.
    - Навигация. Действие кнопки «назад» браузера для сайта, ошибки: 404, 500 и т. д.
    - Инсталляция. Формальное тестирование программы установки (графический интерфейс пользователя, общее удобство пользования, соответствие стандартам), функциональное

тестирование программы установки, тестирование механизма лицензирования и способности противостоять взлому, проверка стабильности работы приложения после установки.

- Нагрузочное. Тесты производительности (количество обрабатываемых запросов в единицу времени), нагрузочные тесты (количество бизнес-пользователей приложения, генерирующих запросы в единицу времени), стресс-тесты (функционирование в условиях экстремальных уровней нагрузки и ограничения ресурсов, а также контроль за скоростью восстановления после стресса), объемное тестирование (увеличение объема хранимых, обрабатываемых данных).
- Ресурсные тесты. Тестирование устойчивости в течение длительного времени.
- Юзабилити. Тестирование удобства пользования приложением определяет, соответствует ли приложение потребностям целевой аудитории и отвечает ли оно требованиям пользователя (сколько шагов и времени нужно пользователям для выполнения их задач в приложении, частота возникновения проблем взаимодействия с пользовательским интерфейсом).
- UI/GUI. Тестирование на соответствие стандартам графических интерфейсов, тестирование с различными разрешениями экрана, тестирование в ограниченных условиях, например, в условиях нехватки памяти, тестирование локализованных версий: точность перевода, проверка длины названий элементов интерфейса, тестирование графического интерфейса пользователя на целевых устройствах.

- Локализация. Проверка правильности перевода согласно тематике данного сайта или программы. Проверка перевода раздела «Помощь» и сопроводительной документации.
- Безопасность. Тестирование безопасности представляет собой ряд услуг, от разработки политики безопасности до тестирования безопасности на уровне приложения, операционной системы и сетевой безопасности. Учитываются три главных критерия: конфиденциальность, целостность, доступность.
- Совместимость. Проверяется функционирование на различных операционных системах, платформах разработки, в различных браузерах.
- Конфигурация. Тестируется функционирование программных систем на различных аппаратных платформах и при различных конфигурациях аппаратно-зависимого программного обеспечения (драйвера, взаимодействие с периферийными устройствами).
- Документация. Тестируется на этапе разработки требований к программному продукту после создания функциональных спецификаций. Помогает избежать логических дефектов и ненужных изменений в продукте до начала его фактической разработки. Позволяет улучшить пользовательскую документацию.
- Прототип. Основная цель тестирования прототипа – выявить потенциальные проблемы в приложении, проверить, насколько приложение соответствует потребностям и ожиданиям пользователя, и обнаружить расхождения с требованиями к графическому интерфейсу пользователя. Проверяется:

структура приложения, формы пользовательского интерфейса, прототип бизнес-логики, логические связи между модулями, навигация, графический интерфейс пользователя;

- субъекту, степени автоматизации, признаку позитивности, времени проведения. Данная группа видов тестирования характеризует как выбор объекта тестирования и роль участника команды разработки, так и особенности условий реализации программной системы и жизненного цикла разработки:
    - Субъект тестирования – лицо или группа специалистов, проводящая тестирование (штатные работники, потенциальные заказчики на стороне разработчика, потенциальные пользователи готовой версии продукта с ограничениями).
    - По степени автоматизации: ручное тестирование, автоматизированное тестирование, полуавтоматизированное тестирование.
    - По признаку позитивности. Позитивное тестирование – это тестирование на данных или сценариях, которые соответствуют нормальному (штатному, ожидаемому) поведению системы. Основной целью позитивного тестирования является проверка того, что при помощи системы можно делать то, для чего она создавалась. Негативное тестирование – это тестирование на данных или сценариях, которые соответствуют нештатному поведению тестируемой системы – различные сообщения об ошибках, исключительные ситуации, запредельные состояния. Основной целью негативного тестирования является проверка устойчивости системы к воздействиям различного рода, валидация неверного набора данных, проверка обработки исключительных ситуаций

(как в реализации самих программных алгоритмов, так и в логике бизнес-правил).

- По времени проведения. Альфа-тестирование проводится на ранней стадии разработки продукта, но в некоторых случаях может применяться для законченного продукта в качестве внутреннего приемочного тестирования. Иногда альфа-тестирование выполняется под отладчиком или с использованием окружения, которое помогает быстро выявлять найденные ошибки. Тестирование новой функциональности проверяет корректность реализации новых задач. Тестирование при приемке (Smoke-testing, дымное тестирование) – поверхностная проверка всех модулей приложения на предмет работоспособности и наличия быстро находимых критических и блокирующих дефектов. Тестирование сборки направлено на определение соответствия выпущенной версии критериям качества для начала тестирования. Приемочное тестирование – формальный процесс тестирования, который проверяет соответствие системы требованиям и проводится с целью определения, удовлетворяет ли система приемочным критериям; вынесения решения заказчиком или другим уполномоченным лицом о приемке приложения. Регрессионное тестирование – это вид тестирования, направленный на проверку изменений, сделанных в приложении или окружающей среде (починка дефекта, слияние кода, миграция на другую операционную систему, базу данных, веб-сервер или сервер приложения), для подтверждения того факта, что существующая ранее функциональность работает, как и

прежде. Регрессионными могут быть как функциональные, так и нефункциональные тесты.

Задания к лабораторной работе

1. Выбрать ранее разработанный программный проект.
2. Провести анализ и составить отчет со следующей структурой:
    a. Цель работы.
    b. Описание программного проекта.
    c. По каждому из элементов классификации, описанному в разделе методических рекомендаций, привести список видов, типов, методов, уровней тестирования, применимых для выбранного проекта.
    d. Привести примеры дефектов, характерных для каждого вида тестирования.
    e. Выводы по работе.
    f. Список использованных источников.
3. Оформить и защитить отчет.

Контрольные вопросы

1. Определяется ли качество ПО качеством программного кода?
2. Какие существуют виды тестирования?
3. Какие существуют типы тестирования?
4. Какие существуют методы тестирования?
5. Какие существуют уровни тестирования?

ЛАБОРАТОРНАЯ РАБОТА №3

Тема лабораторной работы: создание тестовой документации.

Общие сведения по работе

Основным объектом изучения в данной работе являются наборы тестовых сценариев (тест-кейсов). Для удобства управления процессом тестирования и повышения структурированности документации применяют группировку тестовых сценариев в тест-сьюты. Важным является вопрос объема тестовой документации. Необходимо опираться на следующие аспекты: количества применяемых видов тестирования, используемых методов, примененных практик тест-дизайна, квалификации тестировщика. Общими требованиями к документации являются соответствие актуальным требованиям к разрабатываемому программному обеспечению, высокая степень универсальности описания выполняемых операций и условий для более длительного интервала сохранения актуальности, независимости тест-кейсов от других.

Методические рекомендации и материалы

К тестовой документации относятся: план тестирования (тест-план), набор тестов (тест-сьют), тестовый случай (тест-кейс), отчет о дефектах (баг-репорт). Тест-план описывает следующие ключевые элементы процесса тестирования:

- что надо тестировать,
- режимы тестирования,
- график тестирования,
- критерии начала тестирования,
- критерии окончания тестирования,
- окружение тестируемой системы,
- необходимое оборудование и программные средства,

– возможные риски и пути их решения.

Более детально требования к содержанию плана тестирования изложены на рисунке 3.

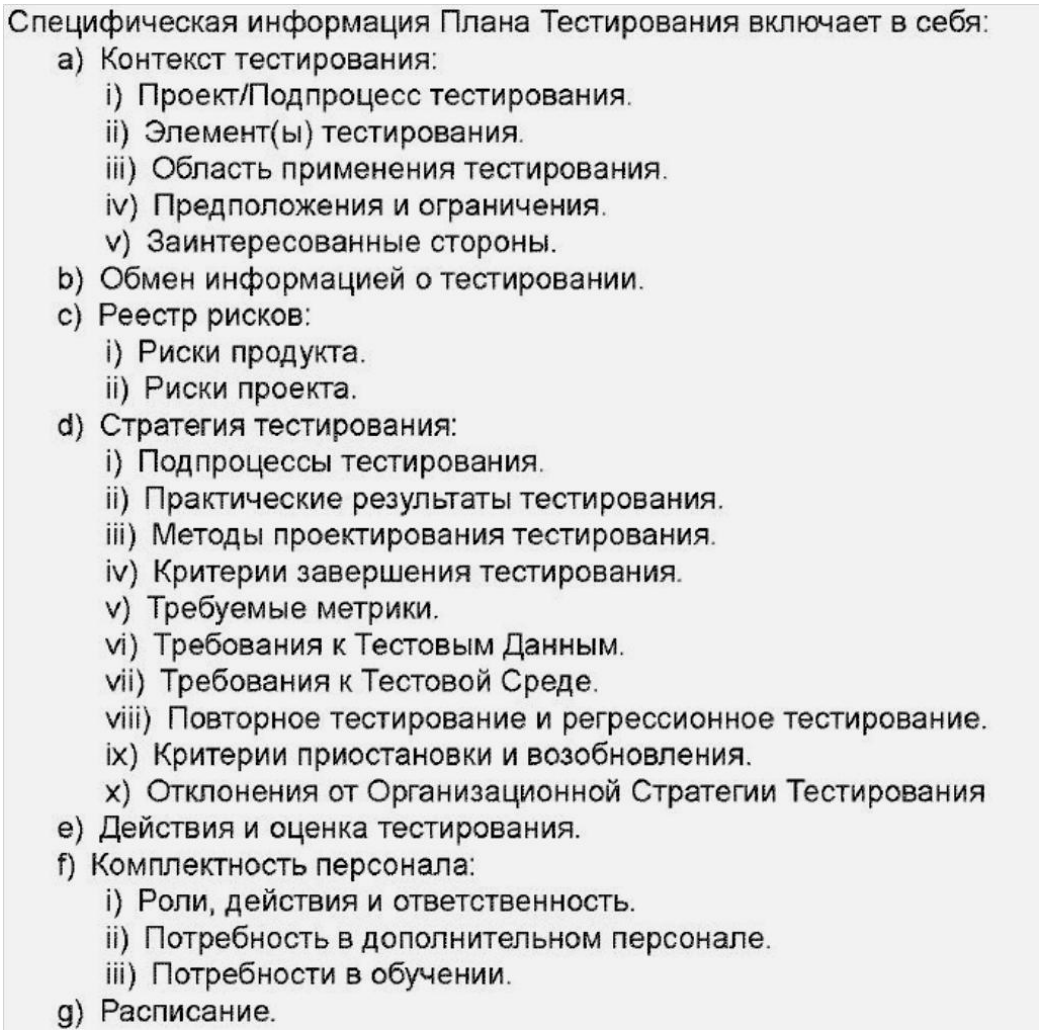
Специфическая информация Плана Тестирования включает в себя:
a) Контекст тестирования:
i) Проект/Подпроцесс тестирования.
ii) Элемент(ы) тестирования.
iii) Область применения тестирования.
iv) Предположения и ограничения.
v) Заинтересованные стороны.
b) Обмен информацией о тестировании.
c) Реестр рисков:
i) Риски продукта.
ii) Риски проекта.
d) Стратегия тестирования:
i) Подпроцессы тестирования.
ii) Практические результаты тестирования.
iii) Методы проектирования тестирования.
iv) Критерии завершения тестирования.
v) Требуемые метрики.
vi) Требования к Тестовым Данным.
vii) Требования к Тестовой Среде.
viii) Повторное тестирование и регрессионное тестирование.
ix) Критерии приостановки и возобновления.
x) Отклонения от Организационной Стратегии Тестирования
e) Действия и оценка тестирования.
f) Комплектность персонала:
i) Роли, действия и ответственность.
ii) Потребность в дополнительном персонале.
iii) Потребности в обучении.
g) Расписание.

Рис. 3. Структура плана тестирования по ГОСТ Р 56922—2016/ISO/IEC/IEEE 29119- 3:2013

Атрибутами тест-плана являются:

– Заголовок / версия / Автор,

– Техническое задание на продукт или иная документация,

– Задачи / функциональность, которая должна быть протестирована,

– Виды проводимого тестирования,

– Список тестовой документации (тест-кейсы, тест-сьюты),

– Список инструментов тестирования,

– Сервер (или иное расположение тестируемой программной системы),
– Ответственные лица (ФИО/ Должность / занятие),
– График тестирования,
– Оценка риска,
– Примечание.

Тест-сьют – группа тест-кейсов для проверки отдельного модуля или функциональности. Атрибутами тест-сьюта являются:

– ID – идентификатор (уникальный в рамках компании),
– Автор,
– Приоритет выполнения,
– Краткое описание,
– Список тест-кейсов.

Пример тест-сьюта для приложения интернет-магазина:

– TS #1: “Модуль авторизации”
    - TC #1.1: “Авторизация пользователя”
    - TC #1.2: “Восстановление пароля”
    - TC #1.3: “Помощь”
    - TC #1.4: …
– TS #2: “Регистрация пользователя”
    - TC #2.1: ...

Тестовый случай (Test Case) – совокупность шагов, конкретных условий и параметров, необходимых для проверки реализации тестируемой функции или ее части. Атрибуты тест-кейса:

– ID.
– Дата.
– Автор.

– **Заголовок / описание.** Список действий, переводящих систему из одного состояния в другое, для получения результата, на основании которого можно сделать вывод об удовлетворении реализации поставленным требованиям.

– **Выполняемые шаги.**

– **Приоритет выполнения.**

– **Предусловия.** Список действий, которые приводят систему к состоянию, пригодному для проведения основной проверки. Либо список условий, выполнение которых говорит о том, что система находится в пригодном для проведения основного теста состоянии.

– Постусловия. Список действий, переводящих систему в первоначальное состояние.

– Дополнительные сведения.

Порядок создания тестовой документации определяется условиями реализации программных систем. Если к началу разработки существовала документация, описывающая требования, то создание тестовой документации должно опираться на эти требования и начинаться непосредственно после их выявления. Если программное обеспечение передается на доработку без какой-либо документации, или требования формализованы не полностью, то создание тестовой документации может быть проведено в режиме исследования.

Далее, применяя одну из техник тест-дизайна, нужно создать наборы тест-кейсов и сгруппировать их в тест-сьюты. Стоит контролировать заполнение обязательных атрибутов тестовой документации. Выполнение лабораторной работы не предусматривает использование какого-либо специального программного обеспечения для учета документации, будет достаточно любого текстового редактора. Но системы ведения тестовой документации могут значительно сокращать трудоемкость для

тестировщика, поскольку обладают возможностями ведения нескольких типов тестовой документации и выполнения манипуляций с ней в рамках процедур тестирования. В качестве примера можно привести сервисы и ПО для тестовой документации: TestLink, TestRail, QA Touch. Примеры тестовой документации приведены в таблицах 1 и 2.

Таблица 1. Примеры оформления списка тест-сьютов

ID	Автор	Приоритет	Заголовок	Список тест-кейсов
1	user	1	Модуль пользователя	1.1 Авторизация 1.2 Регистрация 1.3 Редактирование профиля
2	user	1	Модуль товаров	2.1 Получение новинок 2.2 Получение товаров в категории 2.3 Получение товара
3	user	1	Модуль корзины	3.1 Добавление товара в корзину 3.2 Увеличение количества товара в корзине 3.3 Уменьшение количества товара в корзине
4	user	1	Модуль оформления заказа	4.1 Оформление заказа 4.2 Оформление заказа 4.3 Оформление заказа

Таблица 2. Примеры оформления списка тест-кейсов

ID	Описание (Тип)	Предусловия	Шаги	Ожидаемый результат
1.1	Авторизация (позитивный)	1. Пользователь находится на странице входа в личный кабинет 2. Пользователь был ранее зарегистрирован в системе	1. Ввести в поля «Логин» и «Пароль» логин и пароль пользователя 2. Нажать кнопку «Войти»	1. Пользователь перенаправляется на страницу личного кабинета
1.2	Регистрация (негативный)	1. Пользователь находится на странице регистрации 2. Пользователь был ранее зарегистрирован в системе	1. Ввести в обязательные поля «Имя», «Фамилия», «Пароль», «Подтвердите пароль», «Электронная почта» данные. Также можно ввести данные в поля «Номер телефона» и «Адрес». 2. Нажать кнопку «Зарегистрироваться»	1. Пользователь на странице регистрации получает сообщение «Пользователь с таким email существует»

Продолжение таблицы 2

ID	Описание (Тип)	Предусловия	Шаги	Ожидаемый результат
1.3	Редактирование профиля (негативный)	1. Пользователь находится на странице личного кабинета	1. Нажать на пункт «Профиль» 2. Получить страницу редактирования профиля 3. Нажать на кнопку «Изменить» 4. Ввести в поля «Пароль» и «Подтвердите пароль» разные значения 5. Нажать на кнопку «Сохранить»	1. Пользователю отображается сообщение «Пароли не совпадают»
2.1	Получение новинок (Позитивный)	1. Пользователь находится на одной из страниц магазина	1. Пользователь нажимает на название интернет-магазина, расположенного в шапке сайта	1. Пользователю отображается главная страница магазина со списком новинок
2.2	Получение товаров в категории (Позитивный)	1. Пользователь находится на одной из следующих страниц: на главной странице, на странице «Корзина», на странице входа в личный кабинет, на странице регистрации, на странице каталога товаров, на странице товара	1. Нажать на пункт меню категорий	1. Пользователь перенаправляется на страницу категорий 2. На странице категорий отображается заголовок, совпадающий с названием ранее выбранного пункта меню
2.3	Получение товара (Позитивный)	1. Пользователь находится на одной из следующих страниц: на главной странице, на странице «Корзина», на странице категорий 2. На данной странице отображается товар	1. Нажать на кнопку «Товар»	1. Пользователь перенаправляется на страницу товара 2. Информация о товаре на странице товара должна совпадать с информацией на странице, с которой был осуществлен переход

Окончание таблицы 2

ID	Описание (Тип)	Шаги	Ожидаемый результат	Ожидаемый результат
3.1	Добавление товара в корзину (Позитивный)	1. Пользователь находится на одной из следующих страниц: на главной странице, на странице категорий, на странице товара 2. На данной странице отображается товар	1. Нажать на кнопку «Добавить в корзину»	1. Страница, на которой находится пользователь, обновляется 2. Пользователю отображается сообщение «Товар в корзину успешно добавлен» 3. При переходе на страницу «Корзина» отображается добавленный товар
3.2	Увеличение количества товара в корзине (Позитивный)	1. Пользователь находится на странице «Корзина»	1. Нажать на кнопку «+»	1. В колонке «Кол-во» в строке товара увеличивается значение на 1
3.3	Уменьшение количества товара в корзине (Негативный)	1. Пользователь находится на странице «Корзина» 2. Значение в колонке «Кол-во» в строке товара равно 1	1. Нажать на кнопку «-»	1. Значение в колонке «Кол-во» в строке товара не изменяется

Задания к лабораторной работе

1. Выбрать веб-приложение для тестирования, согласовать с преподавателем.
2. Сформировать отчет с тестовой документацией: список тест-кейсов и тест-сьютов в соответствии с требованиями к атрибутам документации. Не нужно рассматривать функции авторизации / регистрации, поскольку они одинаковые для программных систем.
3. Требования к наличию тестов: smoke-тесты, тестирование навигации, тестирование ввода данных (как минимум две формы),

тестирование бизнес-логики. Обязательно сделать как позитивные, так и негативные тест-кейсы.

4. В отчет по лабораторной работе включить:
    a. Цель работы.
    b. Описание тестируемого приложения.
    c. Тестовую документацию.
    d. Выводы по работе.
    e. Список использованных источников.
5. Оформить и защитить отчет.

Контрольные вопросы

1. Приведите пример негативных тест-кейсов для трех видов тестирования.
2. Перечислите требования к тест-плану.
3. Перечислите требования к тест-сьютам.
4. Перечислите требования к тест-кейсам.
5. Какова связь этапа жизненного цикла разработки программного обеспечения и вида тестовой документации?

ЛАБОРАТОРНАЯ РАБОТА №4

Тема лабораторной работы: методы тест-дизайна.

Общие сведения по работе

Методы тест-дизайна упорядочивают деятельность по проектированию и созданию необходимых наборов тестов для контроля за качеством реализации программных продуктов. Основная задача при этом – минимизация количества тестов и количества их прогонов при максимизации контроля в условиях ограниченных временных, вычислительных и человеческих ресурсов. Частично требования к тест-дизайну изложены в требованиях к тестовой документации, стандартах процессов тестирования, требованиях к использованию инструментов тестирования, а также в описаниях этапов жизненного цикла разработки выбранной модели. В данной лабораторной работе необходимо сосредоточиться на задаче проектирования необходимого набора тест-кейсов для тестирования отдельной функциональности приложений, основываясь на анализе входной информации.

Методические рекомендации и материалы

Одной из проблем при создании набора тестов является разнообразие входных данных. Данные могут подавать на вход программной системы в разном сочетании, в разном объеме. Поэтому проектирование тестов должно учитывать и такие особенности.

Для минимизации выполняемых прогонов тестов применяют техники анализа эквивалентных классов, граничных значений. Сутью подходов является разделение всех подаваемых на вход данных на отдельные группы. Вся совокупность данных в пределах одной группы приводит к одной и той же реакции системы, в то время как данные из разных групп позволяют получить разный результат. Так, например, если тестируются

поля ввода текста и установлено ограничение на минимально допустимую длину текста, равное 5 символам, то ввод строк длиной 1,2,3 или 4 символов должен приводить к одной реакции системы – выводу ошибки. Таким образом, эквивалентный класс – это набор данных или тестов, проверяющих один из вариантов реализации логики функционирования программной системы. Минимизация тестов, выполняемых над системой, достигается за счет использования только одного из значений из эквивалентного класса.

Другая проблема, которая требует рассмотрения – это тестирование поведения системы в граничных значениях эквивалентных классов. Обосновывается эта проблема необходимостью контролировать покрытие всех возможных вариантов условными операторами (например, проверятся код: if (a.equals(“some_string”) в случаях a = null, а не содержит строку “some_string”)), вариантов использования операторов сравнения (x > y или x ≥ y ?) в программном коде. Способом проконтролировать поведение программной системы в таких случаях является анализ эквивалентных классов и граничных между ними значений. Методика тестирования будет заключаться в определении отношения граничных значений к определенному эквивалентному классу. После этого производится выполнение тестов на значениях каждого класса, плюс выполняются прогоны тестов на граничных значениях и в некоторой близкой области к граничным значениям. Например, необходимо протестировать ввод целочисленного положительного числа при условии, что поведение алгоритма меняется при значении переменной, равном 7. Эквивалентные классы будут включать два интервала: [0, …, 7] и (7, …, ∞); При использовании методики тестирования граничных значений нужно выполнить тесты при значении переменной, равном:

– значению из интервала [0, …, 7], например 3,

– значению переменной, равном значению на границе классов, т. е. 7,

– значениям переменной в области близкой к граничному значению (т.к. число целое, то минимальный сдвиг от границы может дать значения, равные 6 и 8),

– значению из интервала (7, …, ∞), например 9.

Методика тестирования предполагает использование всех перечисленных случаев, хотя очевидно, что значения 8 и 9 принадлежат одному классу и могут быть сведены только к одному значению. Использовать или не использовать все указанные значения решают на основе данных: если сдвиг от граничных значений и выбор дополнительных значений эквивалентных классов помогут повысить вероятность локализации дефектов, то нужно использовать все изложенные варианты данных. В случае целочисленных данных, как в примере, сложно сказать, что поведение системы может измениться, поэтому список данных, используемых в тестировании, можно сократить.

Задания к лабораторной работе

1. Для одной из форм приложения выделить эквивалентные классы.
2. Сделать расчет количества тестов для проверки формы приложения с учетом требования минимизации количества проводимых тестов.
3. В отчет по лабораторной работе включить:
    a. Цель работы.
    b. Список используемых тест-кейсов.
    c. Описание эквивалентных классов.
    d. Расчет количества тестов.
    e. Выводы по работе.
    f. Список использованных источников.
4. Оформить и защитить отчет.

Контрольные вопросы

1. Опишите методику выделения эквивалентных классов.
2. В чем цель тестирования граничных значений?
3. Что такое методика черного ящика?
4. В чем разница между методикой черного, белого и серого ящиков?
5. Что представляет собой тест-дизайн?

ЛАБОРАТОРНАЯ РАБОТА №5

Тема лабораторной работы: ручное тестирование

Общие сведения по работе

Работа предусматривает использование ранее созданной тестовой документации в тест-кейсах и тест-сьютах для контроля реализованной функциональности в программной системе. Ручное выполнение тест-кейсов может быть выполнено в зависимости от стадии разработки программного обеспечения (необходимая функциональность должна быть реализована) и является наиболее простым способом реализации процедуры тестирования как таковой. Другие подходы в виде реализации модульных тестов, реализации автотестов могут быть более трудоемкими и менее гибкими.

Методические рекомендации и материалы

Ручное функциональное тестирования подразумевает выполнение тестовых сценариев тестировщиком на специально подготовленном стенде. Порядок проведения ручного тестирования описывается следующими шагами:

1. Анализ исходных документов с требованиями и описанием тестовых случаев.
2. Изучение тест-плана для уточнения сроков, объемов, видов тестирования, ответственных лиц, проводящих тестирование.
3. Выполнение требуемых тест-кейсов с учетом условий выполнения, шагов сценария, эталонных результатов. Ручное выполнение подразумевает взаимодействие тестировщика с элементами пользовательского интерфейса тестируемой программной системы. Эталон необходимо сравнивать с полученным в программной системе результатом.

4. После завершения очередного прогона тест-кейса в отчете по тестированию фиксируются сведения о времени, идентификаторе соответствующего тест-кейса и общем результате: успешно или неуспешно. В случае неуспешного выполнения необходимо описать возникший дефект: на основании шагов и условий тестового сценария нужно описать расхождение в полученном и ожидаемом результате, прикрепить соответствующие дополнительные материалы (скриншоты, видео, логи, дампы) для лучшего понимания разработчиком возникшей проблемы и ускорения поиска путей ее решения.

Ручное выполнение тест-кейсов может быть проведено с использованием соответствующего программного обеспечения, в котором как хранятся сами тест-кейсы, так и организуется создание тест-планов и есть возможность поэлементного просмотра списка тест-кейсов в режиме выполнения.

Описание дефектов производится в соответствии со следующей структурой:

- Краткое описание. Приводится текстовое описание дефекта. Главная цель – пояснить, уточнить особенности дефекта, которые не могут раскрыть остальные поля.
- Серьезность. Отражает объем функциональности, которая не может быть использована из-за возникшего дефекта. Градация из значений – блокирующая (blocker), критическая (critical), значительная (major), незначительная (minor), тривиальная (trivial), предложение к улучшению (enhancement) – позволяет сделать вывод о том, насколько дефект затрагивает работу программной системы и может ли тестирование быть выполнено далее.

- Приоритет является атрибутом, позволяющим управлять очередностью исправления дефектов.
- Шаги к воспроизведению. Строго определенная последовательность действий при взаимодействии с программной системой, приводящая к состоянию, описанному данным дефектом.
- Результат описывает полученное в ходе взаимодействия с системой состояние.
- Ожидаемый результат – некоторое описание эталонного состояния системы, которое ожидалось после выполнения описанных шагов.

Кроме представленных атрибутов описания дефекта можно дополнительно прикладывать материалы, позволяющие точнее воспроизводить дефект и быстрее понимать, в чем состоит проблема.

Задания к лабораторной работе

1. Сформировать по ранее разработанной тестовой документации тест-план, включить в него некоторое подмножество тест-кейсов.
2. Выполнить тест-кейсы, представленные в тест-плане.
3. В отчет по лабораторной работе включить:
    a. Цель работы.
    b. Тест-план.
    c. Отчет по тестированию.
    d. Список выявленных дефектов.
    e. Выводы по работе.
    f. Список использованных источников.
4. Оформить и защитить отчет.

Контрольные вопросы

1. Какие виды тестирования можно выполнять в ручном режиме?
2. Какая документация участвует в ручном тестировании?

3. Может ли существовать программная система, свободная от дефектов?
4. Каковы требования к описанию дефекта?
5. Каков жизненный цикл дефекта?

ЛАБОРАТОРНАЯ РАБОТА №6

Тема лабораторной работы: автоматизация тестирования

Общие сведения по работе

Автоматизированное тестирование является способом повысить скорость контроля качества при разработке программных систем. В данной лабораторной работе необходимо приобрести навыки создания автотестов для контроля функциональности. Работа основывается на разработанной ранее тестовой документации, а именно тест-кейсах, тест-сьютах, планах тестирования. Автоматизация тестирования может работать с различными типами приложений: desktop, мобильные приложения, веб-приложения. Из-за особенностей приложений выбор инструмента будет сильно зависеть от тестируемого приложения. Наибольшее распространение в настоящее время получили инструменты, позволяющие тестировать веб-приложения и мобильные приложения. Это связано с возрастающей популярностью таких типов приложений и возможностями инструментов разработки, преследующих цель повышения кросс-платформенности.

Методические рекомендации и материалы

Автоматизированное тестирование программного обеспечения (Software Automation Testing) – это процесс верификации программного обеспечения, при котором основные функции и шаги теста, такие как запуск, инициализация, выполнение, анализ и выдача результата, выполняются автоматически при помощи инструментов для автоматизированного тестирования. Достоинствами автоматизации тестирования являются:

- Увеличение скорости тестирования без ущерба для результата.
- Возможность выполнять тесты 24/7.

– Уменьшение объема ручных, рутинных, постоянно повторяющихся операций.

– Снижение стоимости итерации тестирования.

– Автоматическое формирование отчетов о тестировании.

– Набор тестов ограничивается производительностью системы, а не доступным резервом времени людей.

В то же время есть ряд недостатков:

– На начальном этапе внедрения автотестов требуется больше времени, чем в сравнении с ручным функциональным тестированием

– Плохо спроектированные/реализованные автотесты ведут к нестабильным результатам и ложным срабатываниям.

– Требуются люди с высокой квалификацией в области тестирования и знанием языков программирования и технологий одновременно.

– Требуется дополнительно время на сопровождение автотестов.

– Дефекты в автотестах ведут к пропуску багов в тестируемом приложении.

Перед началом разработки автоматизированных тестов нужно определить, что нуждается в таком тестировании:

– Труднодоступные места в системе.

– Часто используемая функциональность.

– Рутинные операции.

– Проверка данных, требующих точных математических расчетов.

– Проверка форм с множеством полей.

Не нужно строить автотесты для:

– Постоянно изменяющейся функциональности.

– Случаев необходимого постоянного вмешательства оператора в ходе выполнения автотестов.

– Проверки содержимого изображений, видео ряда, т. д.

Проектирование автоматизированных тестов зависит от поставленной цели. Это может быть нагрузочное тестирование, в котором использование инструментов автоматизации позволяет организовать необходимый режим эксплуатации приложения для снятия показателей производительности и отказоустойчивости, т. е. проверка нефункциональных требований к системе. Другие цели использования автоматизации могут проверять функциональность приложения. Режимами выполнения тестов в таких случаях будут:

– Эмуляция действий пользователя при взаимодействии с элементами пользовательского интерфейса.

– Тестирование программных интерфейсов, если целевая программная система является библиотекой, сервисом или компонентом, способным самостоятельно функционировать и выполнять задачи.

Во всех рассмотренных вариантах автотестов необходимо использовать сценарный подход в качестве основы для скриптов, кода автотестов. Источником будут наборы тест-кейсов.

С технологической точки зрения особенностями процесса создания автотестов являются:

– Автоматизация развертывания тестовых стендов (приложение, СУБД, платформа и т. д.) для выполнения автотестов.

– Написание программного кода автотестов, являющегося или самостоятельным программным проектом или являющимся отдельным модулем тестирования внутри основного приложения.

Обозначенные требования позволяют сделать вывод о необходимости обладания тестировщиком дополнительными компетенциями для создания и проведения автоматизированного тестирования.

После уточнения целей создания автотестов необходимо выбрать инструмент для автоматизированного тестирования. Приведем некоторые примеры:

– Selenium WebDriver является одним из возможных инструментов для проведения функционального тестирования веб-приложений. Его особенностью является использование установленного в операционной системе браузера для выполнения манипуляций с элементами пользовательского интерфейса веб-приложений. Selenium WebDriver является набором абстракций над веб-страницами, позволяющими искать и выбирать необходимые элементы (по имени тега, классу, xpath, …), взаимодействовать с ними (клик, ввод текста, …), осуществлять навигацию по страницами веб-приложения;

– xUnit автоматизация тестирования приложений, созданных на .Net Framework;

– Xamarin UI Tests – тестирование кросс-платформенных мобильных приложений, созданных на платформе Xamarin;

– Xautomation – пакет для linux, позволяющий эмулировать действия пользователя при работе с приложениями, запущенными на ПК;

– Monkey testing – тестирование мобильных приложений путем создания случайных потоков событий взаимодействия с элементами пользовательского интерфейса;

– Apache JMeter – тестирование API, нагрузочное тестирование;

Примеры использовать Selenium WebDriver можно найти в проекте https://git.athene.tech/romanov73/tis-2017. Кроме использования абстракций самого инструмента необходимо создавать автотесты с учетом свойств сопровождаемости, надежности, структурированности. Другими словами, автотесты необходимо создавать так, чтобы требовалось меньше времени на их поддержку и доработку, а случае изменений требований к

программной системе автотесты можно было бы незатратно актуализировать.

Одним из примеров такого подхода является использование POM – PageObjectModel, подход, позволяющий абстрагировать представление тестируемого приложения от сценариев тестирования. Делается это с целью сокращения трудоемкости актуализации тестов в случае изменения верстки веб-страниц приложения, изменения адресации элементов.

Для отработки навыков автоматизации тестирования веб-приложений существует ряд сервисов, предоставляющих типовые веб-компоненты, различные режимы использования. Список таких сервисов:

https://demoqa.com/

http://uitestingplayground.com/

http://the-internet.herokuapp.com/

https://www.globalsqa.com/angularJs-protractor/BankingProject

https://www.saucedemo.com/

https://formy-project.herokuapp.com/

http://automationpractice.com/index.php

Задания к лабораторной работе

1. Основываясь на ранее разработанной тестовой документации, разработать автотесты для проверки функциональности программной системы по варианту в соответствии с требованиями:
    a. Автотесты необходимо проектировать, исходя из свойств сопровождаемости, надежности, структурированности.
    b. Автотесты можно реализовать в форме отдельного приложения в виде модульных тестов.
    c. Автотесты должны эмулировать поведение пользователя при взаимодействии с элементами пользовательского интерфейса.

Обязательно реализовать контроль результатов выполненного сценария.

2. В отчет по лабораторной работе включить:
    a. Цель работы.
    b. Описание реализованных автотестов: инструменты, подходы.
    c. Код автотестов.
    d. Отчет о тестировании (выполненные тест-кейсы, результат, выявленные дефекты).
    e. Выводы по работе.
    f. Список использованных источников.
3. Оформить и защитить отчет.

Контрольные вопросы

1. Каковы цели использования автоматизированного тестирования?
2. В каких случаях можно использовать автоматизированное тестирование?
3. Каковы недостатки автоматизированного тестирования?
4. Что такое локатор?
5. Каковы цели нагрузочного тестирования?