БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

ИИТ БГУИР

Программное обеспечение информационных технологий.

Дисциплина: «Тестирование веб-ориентированных приложений»

Отчет

По лабораторной работе №5

«СТРАТЕГИИ И МЕТОДЫ ТЕСТИРОВАНИЯ»

Выполнил студент группы 481064 Сорока А.А.

Проверил преподаватель: Калитеня И. Л.

Минск 2017 г

**Цель:**

1. Проанализировать бизнес – процесс и произвести его тестирование.

**Задание:**

1. Произвести тестирование бизнес – процесса и дать описание методу тестирования

**Решение:**

1. Основные понятия в тестировании

Тестирование программного обеспечения (Software Testing) – это проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным образом [IEEE Guide to Software Engineering Body of Knowledge, SWEBOK, 2004].

В более широком смысле, тестирование – это одна из техник контроля качества, включающая в себя действия по планированию работ (Test Management), проектированию тестов (Test Design), выполнению тестирования (Test Execution) и анализу полученных результатов (Test Analysis).

Верификация (Verification) – это процесс оценки системы или её компонент с целью определения соответствия результатов текущего этапа разработки условиям, сформированным в начале этого этапа [IEEE].

Валидация (Validation) – это определение соответствия разрабатываемого ПО ожиданиям и потребностям пользователя, требованиям к системе [BS7925-1].

План Тестирования (Test Plan) – это документ, описывающий весь объем работ по тестированию, начиная с описания объекта, стратегии, расписания, критериев начала и окончания тестирования, до необходимого в процессе работы оборудования, специальных знаний, а также оценки рисков с вариантами их разрешения.

Тестовый случай (Test Case) – это алгоритм, описывающий совокупность шагов, конкретных условий и параметров, необходимых для проверки реализации тестируемой функции или её части.

Тестовый набор (Test Suite) – это совокупность тест-кейсов, объединенных по некоторому общему признаку. Например, тест-кейсы для проверки функционала компоненты, используемой в разных формах приложения.

Баг / Дефект Репорт (Bug Report) – это документ, описывающий ситуацию или последовательность действий приведшую к некорректной работе объекта тестирования, с указанием причин и ожидаемого результата.

Объект тестирования – это отдельная логическая единица (форма, подсистема) со всеми элементами управления и диалогами.

2. Инструментарий

Для автоматизации тестирования рекомендуется использовать:

Selenium, RobotFramework - инструменты для создания наборов функциональных тестовых скриптов для проверки web-приложений.

Apache JMeter - инструмент для проведения нагрузочного тестирования и снятия количественных показателей нагрузки на сервер.

ZeNmap, XSpider, Metasploit - сканеры безопасности для исследования топологии сети, поиска уязвимостей в сетевой инфраструктуре приложения и их эксплуатации.

Acunetics WVS, инструментарий OWASP LiveCD - сканер и анализатор логики работы web-приложений, поиска уязвимостей в приложении, и инструменты для тестирования web-приложений.

При ручном тестировании приложений рекомендуется использовать вспомогательные инструменты для исследования приложения:

Snagit - для снятия скриншотов и записи действий на экране.

FireBug - плагин для Firefox, который позволяет отслеживать ошибки приложения на стороне клиента и исследовать (http://firebug.ru/).

Web Developer - плагин для Firefox, инструмент для изменения "на лету" внешнего вида приложения, работы с cookies, css, формами, фреймами, их исходными кодами и т.п. (http://ktonanovenkogo.ru/vokrug-da-okolo/programs/rasshireniya-firefox-opera-web-developer-dlya-firefox.html).

Intercepter-NG, WinDump и др. – снифферы для перехвата и анализа сетевого траффика.

3. Категории тестов

Рекомендуется последовательно разрабатывать отдельные тест-комплекты следующих категорий (подробнее читайте на http://www.protesting.ru/testing/testtypes.html):

Init-тесты инициализации и авторизации (init / login / logout).

Обязательные действия в данной категории:

- для основного Init-теста в тестовом наборе добавить используемые во всех тестах глобальные переменные;

- для основного Init-теста в тестовом наборе добавить проверку корректности формы авторизации (все элементы формы присутствуют и доступны);

- для основного Init-теста в тестовом наборе добавить проверку того, что все обязательные элементы интерфейса главной формы приложения присутствуют и доступны;

- Init-тесты (шаблоны) для всех остальных тестов, разрабатываемые перед непосредственной реализацией основных проверок, должны содержать блок команд для корректного входа и выхода из приложения.

Функциональные CRUD-тесты (Create, Read, Update, Delete) для тестирования интерфейса стандартных операций по созданию, чтению, изменению, удалению элементов web-формы, добавляются в соответствующие Init-тесты.

Обязательные действия в данной категории:

- добавить проверки возможности добавления нескольких разнотипных элементов, из различных классов тестовых данных;

- добавить проверки возможности редактирования добавленных данных;

- добавить проверки возможности удаления данных;

- добавить проверки успешного обновления данных.

Функциональные регрессионные тесты для проверки корректности решенных в системе bug/task-трекинга задач и отсутствия ранее имеющихся ошибок.

Обязательные действия в данной категории:

- добавить проверки важных задач task-ов (имеющих критичный функционал или влияющих на логику работы приложения);

- добавить проверки всех задач bug-ов, поставленных в системе bug/task-трекинга.

Функциональные позитивные тесты для проверки корректности завершения разрешенных (валидных) операций, входящих в CRUD-тесты (например, проверка того, что после сохранения корректно введенной информации выводится сообщение об успехе операции), добавляются в соответствующие Init-тесты.Обязательные действия в данной категории:

- добавить проверки наличия ключевых элементов формы (все кнопки и надписи на своих местах);- добавить проверки фактического выполнения операций CRUD: добавляемые элементы действительно добавились, после редактирования данные изменились, после удаления данные действительно удалены, после обновления имеющиеся на форме данные остаются неизменными и т.п.;

- добавить проверки наличия сообщений об успехе операций CRUD.

Функциональные негативные тесты для проверки корректной обработки неправильно введенных данных (например, проверка того, что при попытке ввода в текстовое поле слишком длинной строки выводится сообщение об ошибке), добавляются в соответствующие Init-тесты.

Обязательные действия в данной категории:

- добавить проверки корректности обработки недопустимых или некорректных данных для операций CRUD: при таких данных программа не должна допускать их сохранения, редактирования или обновления в БД, а также не допускать удаления данных, с которыми имеются связи, без предварительной обработки этих связей и т.п.;

- добавить проверки наличия сообщений о некорректных действиях пользователя или неверных исходных данных.

Функциональные end-to-end сценарии, имитирующие конкретную последовательность действий реального пользователя (например, проверка корректного выполнения последовательности действий по входу в систему, открытию нужной формы, вводу данных в поля этой форму, сохранению данных и выходу из системы), добавляются в соответствующие Init-тесты.

Обязательные действия в данной категории:

- добавить несколько различных вариантов завершенных и логичных действий пользователя при работе с приложением.

Нагрузочные тесты, для проверки возможности "нормальной" и корректной работы в системе определенного числа пользователей выполняющих типичные действия (для имитации работы пользователей возможно использовать, например, уже имеющиеся CRUD-тесты или end-to-end сценарии).

Количественные показатели измеряются для случаев:

a) работы приложения без нагрузки (для определения сравнительных характеристик);

b) работы приложения под равномерно увеличивающейся нагрузкой (с различным числом виртуальных пользователей или различным числом подключений к узлам приложения);

c) работы приложения под пиковой нагрузкой (с различным числом виртуальных пользователей или различным числом подключений к узлам приложения).

Обязательные действия в данной категории:

- определить цели и объекты нагрузки;

- выбрать подходы и методики тестирования нагрузки;

- определить количественные и качественные показатели для нагрузочного тестирования (например, измерить среднее время доступа к приложению, данным и основным подсистемам приложения, степень загрузки процессоров, сетевых адаптеров, файла подкачки, памяти серверов приложений и БД).

Тесты информационной безопасности, которые должны включать в себя проверки безопасного (с точки зрения ИБ) функционирования приложения с точки зрения бизнес-логики работы приложения, даже при работе под нагрузкой или попытках анализа приложения на наличие уязвимостей или взлома.

Обязательные проверки в данной категории:

- выполнить проверки на наличие уязвимостей согласно классификации WASC (WASC Threat Classification, http://projects.webappsec.org/w/page/13246978/Threat-Classification).

4. Предполагаемый набор проверок для автотестов

CRUD-тесты:

- добавление. Добавляем в цикле 3-4 элемента.

- редактирование. Редактируем один из созданных элементов.

- обновление. Обновляем данные.

- удаление. Удаляем в цикле все созданные элементы.

- проверка на отсутствие ошибок. После каждого из вышеперечисленных действий, а так же при открытии новых форм и диалогов проверять на отсутствие ошибок.

В CRUD-тестах используем для этого наборы команд assert\*.

Позитивные тесты:

- длина полей. Проверяем, что в поля ввода вводится строка максимально допустимой длины.

- ОДЗ полей. Проверяем область допустимого значения полей. Актуально, например, для математических функций (cos x и тд) или конкретные ограничения (количество вариантов не больше 50 и т.д.)

- открытие выпадающих списков. Проверяем, что комбобоксы и выпадающие списки при нажатии открываются.

- выбор флажков, переключателей. Проверяем, что чекбоксы и радиобатоны выбираются.

- орфография. Попутно проверяем орфографию. В частности, названия полей и столбцов. Для этого, при обращении к элементу можно использовать заголовок таблицы.

- сортировка. Если нет возможности проверить корректность сортировки, то проверить, что после сортировки по всем имеющимся столбцам не выходит ошибка. В позитивных тестах для проверок используем блок команд verify\*.

- модальность. Проверяем на модальность все открывающиеся окна, диалоги, запросы на удаление, выход и т.д.

- активность кнопок. Проверяем, что при заполненных обязательных полях или других необходимых условиях, кнопки активны.

- всплывающие подсказки на кнопках. Проверяем, что при наведении на кнопки и, возможно, другие элементы появляются правильные тултипы.

- добавление, редактирование, удаление. В дополнение к CRUD-тесту, проверяем что эти операции действительно выполняются. То есть, что элемент после сохранения - появляется, на редактирование - открывается, после удаления - пропадает.

Негативные тесты:

- недоступность кнопок. Проверяем, что при невыполнении обязательных условиий, кнопки неактивны.

- длина полей. Проверяем, что в поля ввода не вводятся строки меньше и больше минимальной и максимальной допустимой длины.

- ОДЗ полей. Проверяем, что в поля не вводятся значения, не входящие в область допустимых (см. аналогичный пункт в позитивных тестах).

- ошибки. После ввода некорректных данных проверяем, что выходит ошибка. Это может быть выделение поля красным цветом (проверяем свойства поля), всплывающая подсказка (наводим курсор на требуемое поле), сообщение или объединение нескольких уведомлений об ошибке.

В негативных тестах для проверок используем блок команд verify\*.

- допустимые символы. Проверяем, что в поля с ограниченным набором допустимых символов не вводятся остальные.

- доступность дублирования. Проверяем, что не добавляются дублирующие элементы (если это недопустимо).

- тримминг. Проверяем, что после сохранения убираются пробелы и табы до первого значащего символа и после последнего значащего символа.

- ограничение полей даты. Проверяем, что не вводятся некорректные даты, в частности пятизначное число года, не выбирается прошедшая дата, если это недопустимо.

- недоступость полей. Проверяем, что в недоступные поля ничего нельзя вбить. Для этого проверяем свойства полей.

End-to-end сценарии:

- проверка ролей. Проверяем отдельным тестом каждую роль, участвующую в бизнес-процессе.

4.1. Проверка типичных ошибок

Отображение понятных пользователю сообщений об ошибке при обработке исключений (вместо непонятных системных).

Ограничения на количество символов или длину строки в поле ввода.

Допустимость ввода отрицательных, нулевых, пустых значений по каждому полю.

Наличие сортировки по умолчанию в списках.

Корректность сортировки после добавления новой записи к уже отсортированным.

Допустимость добавления дублирующих записей.

Ограничение поля год (от 0 до 9999 или по договоренности с РП, но не отрицательные).

Возможность удаления изображений и записей с загруженными изображениями.

Корректность скроллинга при большом количестве данных (комбобоксы, таблицы).

Орфографию заголовков полей - в соответствии с ТЗ.

Корректность отображения данных после ухода со вкладки и возвращения на неё (в т.ч. после обновления данных).

Правильную нумерации строк и полей записей.

Область допустимых значений для математических переменных (например, значения для cosφ от -1 до 1).

Допустимые значения для налоговых/финансовых/банковских номеров и сокращений (например, ИНН - не более 12 символов, КПП - не более 9, для остальных - см. википедию).

Проверка на то, что все строки тримятся (после сохранения должны убираться пробелы и табы до первого значащего символа и после последнего значащего символа: " Татарстан " => "Татарстан").

Модальность диалоговых окон.

Блокировку кнопок редактирования/удаления в процессе открытия диалогов редактирования или удаления соответствующих элементов (с целью предотвращения повторного открытия диалогов или удаления).

Если все поля КЛАДРа обязательны к заполнению, то после выбора любого варианта в одном комбобоксе, должен заполняться адресами комбобокс ниже (хотя бы одним значением по умолчанию - прочерком).

5. Процесс тестирования

5.1. Процесс разработки GUI-тестов для проверки логики работы приложения целиком

Составить функциональную схему логики работы приложения: выделить основные типичные формы и сгруппировать их по реализуемому функционалу (справочники, админка, формы ввода данных, формы для выгрузки данных и т.п.).

Создать комплекты шаблонов для тестов требуемых категорий (файлы "пустышки", содержащими заголовки и основную структуру теста).

Разработчик теста выбирает для тестируемой формы соответствующий ей шаблон и заполняет его проверками, требуемыми для конкретной категории теста.

Отдельные тест-кейсы можно объединить в тест-сьюты, исходя из проверяемого ими функционала.

Написать скрипты для запуска тестов.

Настроить автозапуск скриптов на сервере сборки после сборки приложения.

5.2. Процесс разработки GUI-тестов для отдельных задач, поставленных программистам

Начать ручное тестирование задачи, поставленной в системе bug/task-трекинга и имеющей статус Resolved.

Открыть имеющийся или создать новый шаблон теста для проверки функционала элементов приложения, разработанных в рамках этой задачи.

Выполнять ручное тестирование элементов и если они рабочие, то сразу заносить команды для проверки их работоспособности в тест.

Если найдена ошибка, зафиксировать ее в системе bug/task-трекинга (можно приложить скриншот ошибки) и переоткрыть задачу для разработчика.

Ждать исправления ошибок разработчиками и приостановить разработку теста.

После исправления ошибок запустить тест в IDE для тестирования до места ошибки, убедиться в её отсутствии и продолжить разработку теста, дополнять его новыми проверками и т.д.

5.3. Отслеживание процесса разработки тестов в системе bug/task-трекинга

В процессе разработки тестов, с отслеживанием статусов поступаем аналогично процессу разработки приложений:

При разработке (доработке, поправке) теста, ставим у соответствующей задачи в системе bug/task-трекинга статус In progress (кнопка Start progress в Jira).

Разрабатываем тест, включая в него обязательные проверки.

Проверяем тест на работоспособность самостоятельно (путем полной прогонки в IDE для тестирования и на тестовом стенде).

Ставим задаче по разработке теста статус Resolved.

Задача переходит к ведущему тестировщику (проверяющему), назначенному на проверку теста. Он ставит ей статус Testing (кнопка Начать тестирование в Jira).

Тот, кто проверяет тест, обновляется из репозитория и прогоняет обновленный тест локально, а также смотрит на результаты прогонки теста в системе управления тестами.

Если тест работает, останавливаем его разработку (кнопка Stop progress в Jira). Выкладываем тест в репозиторий.

Варианты проверки теста:

- если тест работоспособен, тогда проверяющий ставит задаче по разработке теста статус Tested (кнопка Успешно протестировано в Jira).

- если тест не работоспособен, то проверяющий переоткрывает задачу по разработке теста и ставит ей статус Reopen (кнопка Переоткрыть в Jira).

5.4. Правила смены статусов задач в системе bug/task-трекинга

Групповые задачи в целом не тестируются, но их статус меняется на Testing.

- статус групповой задачи нужно изменить на Tested после смены статуса всех подзадач на Tested;

- если статус любой подзадачи меняется на Reopened, то необходимо изменить статус на Reopened у групповой.

Тестируются только задачи со статусом Resolved и вошедшие в последнюю сборку в системе управления тестами. Задачи, которые связаны с кодированием или управлением проектом - тестирует (отлаживает) и закрывает разработчик или РП.

После тестирования подзадачи нужно:

- или сменить ее статус на Tested (при этом возможно дополнительно создать новую задачу по улучшению или доработке функционала);

- или сменить ее статус на Reopened с добавлением комментария, причины переоткрытия, логов, снимков экрана и пр.

Задачи со статусом Tested могут быть дополнительно проверены перед сменой их статуса на Closed. Статус Closed устанавливает РП, или один из ведущих разработчиков проекта.

Задачи со статусом Closed передаются на сборку релиза.

5.5. Правила расстановки приоритетов для задач-багов

Blocker - ставится задаче в случае, если нельзя выполнить или выполняются с ошибками базовые CRUD-операции.

Critical - ставится задаче в случае, если не реализована или работает некорректно важная для системы или бизнес-процесса логика.

Major - ставится задаче в случае, если не работает логика обычной (не критичной) задачи.

Minor - ставится задаче в случае, если требуются значительные поправки в стилях и интерфейсе (например вообще не видно кнопок, съехавших за экран, но они есть и работают).

Trivial - ставится задаче в случае, если требуются незначительные поправки в стилях и интерфейсе, орфографии и т.п.

6. Подготовка и ведение тестплана

Тест-план в самом простом случае можно формировать в табличном виде в Excel. Этот документ отражает анализ логики работы приложения и содержит его структуру и описание объектов, для которых будут разрабатываться тесты. Табличная организация тест-плана позволяет представить функционал сложного приложения более наглядно, по отдельным логическим модулям, а также упрощает слежение за разработкой тестов, для чего достаточно построить простые гистограммы готовности тестов.

В тест-плане указываются следующие данные:

* имя тестового-набора, например, "CRUD-тесты для проверки базового функционала интерфейса РЖД Энергопаспорт";
* рабочий идентификатор, например, rzdep\_Test\_suite\_CRUD;
* тип тестов в наборе, например, CRUD-тесты;
* комментарий к тестовому набору;
* общая готовность тестового набора в %;
* разбиение приложения на логические модули и подсистемы, например, тесты для подсистемы безопасности и администрирования, тесты для справочников, тесты для различных форм, при необходимости, даже тесты для отдельных подвкладок;

Для каждого теста в отдельной подсистеме указывается:

* имя теста, например, "Основной АРМ, подсистема авторизации";
* готовность теста в %;
* приоритет теста;
* идентификатор теста, например, 0\_1\_00\_000\_Enter\_system\_Main;
* комментарии.