МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«ТЮМЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ НАУК
Кафедра программного обеспечения

РЕКОМЕНДОВАНО К ЗАЩИТЕ В ГЭК
И ПРОВЕРЕНО НА ОБЪЕМ
ЗАИМСТВОВАНИЯ
Заведующий кафедрой
д.п.н., профессор
_____ И.Г. Захарова
29 июня 2018 г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ АВТОМАТИЗАЦИ ПРОЦЕССА ТЕСТИРОВАНИЯ WEB-ПРИЛОЖЕНИЙ

02.04.03. Математическое обеспечение и администрирование информационных систем

Магистерская программа «Высокопроизводительные вычислительные системы»

Выполнила работу
студентка 2 курса
очной формы обучения
Чашкова
Анна
Дмитриевна

Научный руководитель
к.физ-мат.н., доцент
кафедра программного обеспечения
Ступников
Андрей
Анатольевич

Рецензент
к.т.н., доцент кафедры ИС ИМиКН ТюмГУ,
ген. Директор ООО «Минт Рокет»
Карякин
Иван
Юрьевич

Тюмень - 2018

Оглавление

Введение 5

Глава 1. Общие вопросы организации тестирования программного обеспечения 7

1.1. Основные определения 7

1.2. Формы проведения тестирования 8

1.2.1. Модульное тестирование (Unit testing) 8

1.2.2. Системное тестирование (System testing) 8

1.2.3. Интеграционное тестирование (Integration testing) 9

1.2.4. Приемочное тестирование (Acceptance testing) 10

1.3. Виды тестирования 10

1.4. Методы тестирования 12

1.4.1. Метод черного ящика 12

1.4.2. Метод серого ящика 13

1.4.3. Метод белого ящика 14

1.5. Виды тестирования по степени автоматизированности 16

1.5.1. Ручное тестирование 16

1.5.2. Автматизированное тестирование 17

1.5.3. Сравнение ручного и автоматизированного тестирования 18

1.6. Тестирование Web приложений 19

1.6.1. Структура web-приложений 19

1.6.2. Особенности тестирования web-приложений 21

1.7. Инструмент Selenium, как средство для автоматизации тестирования WEB-приложений 23

1.7.1. Selenium WebDriver 23

1.7.2. Selenium RC 24
1.7.3. Selenium Server 25
1.7.4. Selenium Grid 25
1.7.5. Selenium IDE 26
1.8. Вывод по главе: общие вопросы организации тестирования программного обеспечения 26
Глава 2. Разработка алгоритмов для нахождения тестовых наборов 27
2.1. Постановка задачи 28
2.2. Полный перебор 28
2.3. Способ минимальных проверок 31
2.4. Атомарные проверки 33
2.5. Попарные проверки 36
2.6. Оценка сложности алгоритмов 40
2.6.1. Оценка сложности алгоритма полного перебора 40
2.6.2. Оценка сложности алгоритма минимальных проверок 41
2.6.3. Оценка сложности алгоритма атомарных проверок 41
2.6.4. Оценка сложности алгоритма попарных проверок 42
2.7. Сравнение тестовых подходов 43
2.7. Вывод по главе: разработка алгоритмов генерации тестовых наборов 43
Глава 3. Автоматизация тестирования web-приложений 44
3.1. Приложение для генерации тестовых наборов 44
3.1.1. Описание классов приложения для генерации тестовых наборов . 45
3.1.2. Пример работы приложения 47

3.2. Разработка автоматических тестов для тестирования сервисов Customization Loader и Image Creation 49

3.2.1. Customization Loader и Image Creation 49

3.2.2. Создание тестовых наборов для тестирования сервисов Customization Loader и Image Creation 52

3.2.3. Описание классов приложения для проведения автоматического тестирование порталов Customization Loader и Image Creation 55

3.2.4. Автоматические тесты для тестирования сервисов Customization Loader и Image Creation 57

Заключение 59

Список используемой литературы 60

Приложение 1. Нахождение тестового набора с использованием метода полного перебора 63

Приложение 2. Нахождение тестового набора с использованием метода минимальных проверок 64

Приложение 3. Нахождение тестового набора с использованием метода атомарных проверок 65

Приложение 4. Нахождение тестового набора с использованием метода попранных проверок 66

Приложение 5. Проверка корректности всех ссылок на порталах Customization Loader и Image Creation 68

Приложение 6. Создание записи о плагине на портале Customization Loader 69

Введение

Появление персональных компьютеров способствовало разработке программного обеспечения. С каждым годом программных продуктов, используемых людьми для различных целей, становится все больше.

Нефтесервисная компания Schlumberger предоставляет широкий спектр услуг, связанных с разработкой нефтяных месторождений. Компанией разрабатывается огромное количество программного обеспечения, как для нефтяной промышленности, так и просто для удобства работы в компании. За последние пару лет в компании стала активно развиваться web-разработка.

Контроль качества считается важным в процессе разработки программного обеспечения, потому, что обеспечивает безопасность, надёжность и удобство создаваемого продукта. В компании Schlumberger в каждом техцентре существует отдел по обеспечению качества программного обеспечения. В круг обязанностей инженеров по тестированию входит тестирование программного продукта, разработка тест-плана и тестовых сценариев для разрабатываемых программ. Но разработка тестовых наборов порой занимает много времени, особенно, когда есть необходимость вводить новые тесты, добавляя параметры или значения к уже существующим. А тестирование одних и тех же сценариев со временем превращается в монотонную работу, что может привести к пропуску каких-либо ошибок ввиду человеческого фактора.

Актуальным вопросом в компании Schlumberger для инженеров по качеству является большие временные затраты на проектирование тестов и проведения тестирования одних и тех же сценариев неоднократно. Автоматизация процесса создания тестовых наборов и тестирования web-приложений и является целью данной магистерской диссертации.

Задачами выпускной квалификационной работы в связи с указанной целью являются:

1. Изучить методы, подходы, формы и виды тестирования;

2. Изучить тестирование web-приложений и инструмент для их автоматизации Selenium;
3. Формально описать и разработать алгоритмы создания тестовых наборов для следующих подходов: метод полного перебора, метод минимальных проверок, метод атомарных проверок, метод попарных проверок;
4. Разработать и внедрить приложение для генерации тестовых наборов с помощью разработанных алгоритмов;
5. Разработать программное обеспечение для автоматического тестирования web-приложений Customization Loader и Image Creation.

Глава 1. Общие вопросы организации тестирования программного обеспечения

1.1. Основные определения

Одним из основных методов оценки качества программного обеспечения является тестирование. [22]

Тестирование – это деятельность, направленная на оценку качества продукта с целью выявления каких-либо дефектов и проблем. [4]

Введем несколько основных понятий тестирования.

Верификация (Verification) – это процесс оценки системы на соответствие результатов в данный момент с теми, что были определены в начале текущей фазы.

Валидация (Validation) – это процесс определения соответствия разрабатываемого модуля ожиданиям и потребностям пользователя, а также требованиям к системе.

План тестирования (Test plan) – это документ, включающий в себя описание различных объектов и шагов тестирования, стратегии тестирования, критерии начала и окончания, нужное оборудование и оценку рисков с вариантами их разрешения.

Тестовый случай (Test case) – это последовательность шагов, каких-либо условий и параметров, необходимых для проверки тестируемой функции, процедуры или какой-либо иной части тестируемого программного продукта.

Итоговый отчет (Report) – это документ, включающий в себя ситуации и последовательные шаги, которые привели тестируемый модуль к некорректной работе в процессе тестирования. Иногда в данном документе указываются причины некорректного поведения приложения и результат, который должен был получиться в ходе выполнения описанных шагов.

Время прохождения тестовых случаев (Test case pass time) – это время от начала прохождения шагов до получения результата теста. [11, 20]

1.2. Формы проведения тестирования

1.2.1. Модульное тестирование (Unit testing)

Модульное тестирование — это процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы.

Модульные тесты используются при разработке программного обеспечения. Они могут быть созданы как до написания исходного кода, так и после этого. Модульное тестирование представляет собой написание тестов, которые покрывают изменяемое место в программе, после чего пишется код, который должен пройти ранее написанный тест. Благодаря такому подходу разработка приложения становится проще. Иногда модульное тестирование проходит иначе. Изначально создается функционал программы, а затем создаются модульные тесты. Однако практически невозможно достичь 100% покрытия кода программы модульными тестами. [8]

1.2.2. Системное тестирование (System testing)

Системное тестирование представляет собой тестирование программного продукта в том состоянии, в котором тестируемое программное обеспечение будет внедряться в эксплуатацию.

С помощью системного тестирования можно обнаружить ряд дефектов, таких как некорректная работа некоторых функций программы, отсутствие необходимого функционала в системе, нестабильная работа, ошибки взаимодействия с другими системами и ошибки, возникающие при использовании специфических тестовых данных или их комбинаций.

Преимуществами данного тестирования являются:

1. Уменьшение количества дефектов при эксплуатации;
2. Возможность использования тестовых сценариев в качестве обучающих материалов для будущих пользователей системы;
3. Возможность заранее обнаружить какие-либо ошибки, что позволит исправит их перед тем, как продукт попадет в эксплуатацию. [16]

1.2.3. Интеграционное тестирование (Integration testing)

Интеграционное тестирование предназначено для взаимодействия с различными частями системы (оборудованием, операционной системы), а также для проверки связи между различными компонентами программы.

Интеграционное тестирование делится на 2 уровня, на которых проверяется взаимодействие между различными системами:

1. Компонентный интеграционный уровень (Component Integration testing): Проверка происходит после проведения модульного тестирования.
2. Системный интеграционный уровень (System Integration Testing): Проверка происходит после проведения системного тестирования.

Существуют следующие подходы интеграционного тестирования:

1. Снизу вверх (Bottom Up Integration): все процедуры, функции и низкоуровневые модули собираются вместе и затем происходит тестирование. После, собирается следующий уровень модулей для проведения этого же теста. Обычно данное тестирование проводится, когда уже все или почти все модули готовы. В таком случае интеграционное тестирование позволяет определить степень готовности приложения.
2. Сверху вниз (Top Down Integration): при данном подходе сначала тестируются все высокоуровневые модули, и затем, в ходе тестирования постепенно добавляются модули более низкого уровня. Все низкоуровневые модули изначально симулируются заглушками с аналогичной функциональностью, а затем, когда модули готовы, они заменяются реальными активными компонентами.
3. Большой взрыв ("Big Bang" Integration): всё или практически все разработанные модули собираются вместе в виде законченной системы или ее основной части, и затем проводится интеграционное тестирование. [10]

1.2.4. Приемочное тестирование (Acceptance testing)

Приемочное тестирование необходимо для определения степени готовности программного продукта к эксплуатации. Данное тестирование проводится с помощью набора тестовых сценариев.

Данное тестирование могут проводить как специалисты, так и конечные пользователи. Чтобы качество проверки и надежность проводимых испытаний разработанного программного продукта были выше, а само тестирование было произведено в установленные сроки, то привлечение опытных тестировщиков значительно упростит весь этот процесс.

Приемочное тестирование проходит в 3 этапа: подготовка, проведение, отчет. Подготовка заключается в изучении продукта тестирования, разработке программ и методик испытаний и подготовку приемочных тестов. Проведение приемочного тестирования включает в себя тестирование продукта на соответствие всем правилам с помощью автоматических тестов, разработанных программ и методик, а также функциональное тестирование, которое проводится специалистами. Затем, по завершению предыдущих двух этапов составляется отчет с перечнем ошибок, которые необходимо устранить перед тем, как разработанный программный продукт попадет к конечному пользователю. [1]

1.3. Виды тестирования

Функциональное тестирование (Functional testing) – это тестирование, которое проводится с целью убедиться, что все функциональные требования действительно реализованы.

Тестирование производительности (Performance testing) нацелено на проверку надежности, сколько ресурсов потребляет программа, масштабируемости и насколько быстро она работает. [15]

Стрессовое тестирование (Stress testing) обычно применяется для определения пределов пропускной способности программного продукта, надёжности системы во время диспропорциональных или экстремальных

нагрузок и предоставляет информацию о достаточной производительности системы в случае, если текущая нагрузка сильно превысит ожидаемый максимум. [1]

Нагрузочное тестирование (Load testing) проводится с целью оценить поведение приложения под ожидаемой нагрузкой.

Тестирование стабильности (Stability testing) проводится для того, чтобы убедиться в том, что программный продукт в течении длительного времени выдерживает ожидаемую нагрузку.

Тестирование удобства использования (Usability testing) - это тестирование, направленное на определение степени удобства обучаемости, использования, привлекательности и понятности для пользователей разрабатываемого продукта. По отношению к готовому продукту оно может проходить посредством тестирования черного ящика (black box testing), а в процессе разработки программного продукта путем тестирование белого ящика (white box testing). При проведении тестирования путем белого ящика проверяются удобства использования внутренних классов, объектов, переменных и методов, а также рассматривается удобство изменения, расширения системы и интеграции ее с другими модулями или системами.

Тестирование пользовательского интерфейса (UI testing) проводится с целью убедиться, что он соответствует утвержденным стандартам и их требованиям. При данном тестировании проверяются такие моменты, как ввод с клавиатуры, вставка скопированных значений в ячейки, как ведет себя приложение при изменении размеров окна, имеет ли приложение уникальную иконку и как отображаются такие элементы как кнопки, меню, списки, текст.

Тестирование безопасности (Security testing) - это оценка уязвимости программного продукта к различным атакам. Данное тестирование проверяет реакцию защитных механизмов, встроенных в систему, на какое-либо проникновение. При тестировании безопасности испытатель играет роль хакера, поэтому ему дозволены любые действия по отношению к тестируемому программному обеспечению.

Тестирование локализации (Localization testing) подразумевает проверку множества различных аспектов, связанных с адаптацией разработанного программного обеспечения для пользователей из других стран. К примеру, данное тестирование проверяет как поведет себя система для иностранных пользователей, если этот пользователь на каком-либо сайте введет текст символами, которые не распознает данная программа.

Тестирование совместимости (Compatibility testing) — это вид нефункционального тестирования, целью которого является проверка корректной работы продукта в определенном окружении. [15]

Таким образом, из вышесказанного видно, что видов и форм тестирования очень много. Многие из них применяются в компании Schlumberger при проведении тестирования программных продуктов, разрабатываемых в компании, а именно: компонентное тестирование, модульное, функциональное, тестирование производительности, стабильности, удобства, пользовательского интерфейса и безопасности.

1.4. Методы тестирования

1.4.1. Метод черного ящика

Тестирование методом черного ящика производится лишь по функциональной спецификации и требованиям. При данном тестировании доступа к коду тестируемого продукта и какой-либо базе данных, нет. Но при проведении тестирования данным методом известно, каким должен быть результат при конкретных данных, которые подаются на вход. Данное тестирование проходит так, как с данным программным обеспечением работал бы обычный пользователь, который не имеет никакого отношения к разработке данного программного продукта.

При тестировании методом черного ящика тестировщик придерживается следующих шагов:

1) Изучение спецификаций и требований;
2) Выбор входных значений;

3) Определение ожидаемых выходных значений Входные значения;
4) Исполнение тестов;
5) Сравнение полученных результатов с ожидаемыми

Число тестов определяется числом входов и диапазоном возможных значений входных параметров. Перебор всех возможных входных параметров, как правило, невозможен.

Преимущества тестирования методом черного ящика:

- Тестирование с точки зрения пользователя;
- Не требует специальных знаний (например, конкретного языка программирования);
- Позволяет найти проблемы в спецификациях;
- Можно создавать тесты параллельно с кодом;
- Тестировщик может быть отделен от разработчиков.

Недостатки тестирования методом черного ящика:

- Эффективность зависит от выбора конкретных тестовых значений;
- Необходимость наличия четких и полных спецификаций;
- Невозможность сконцентрироваться на особо сложных частях кода;
- Трудность локализации причины дефекта;
- Возможность не протестировать часть кода. [18]

1.4.2. Метод серого ящика

Тестирование методом серого ящика – это тестирование, при котором устройство программы известно лишь частично. Например, тестировщик имеет доступ к базе данных или внутренней структуре и алгоритмам работы программного обеспечения для написания максимально эффективных тест-кейсов. Метод серого ящика – это комбинация черного и белого ящиков:

- Тестировщик знает частично или полностью внутреннее устройство тестируемого объекта;
- Тестировщик находится на уровне пользователя

При тестировании методом серого ящика знаниями для тестировщика служат не только личный опыт, но документация, где содержится вся необходимая информация о работе и реализации функционала тестируемого программного продукта.

Преимуществами тестирования методом серого ящика являются:

- Метод включает в себя ряд преимуществ тестирования методами черного и белого ящиков;
- Метод серого ящика не базируется на исходном коде и двоичных файлах, а основан на функциональной спецификации и архитектурном построении приложения;
- Тестировщик сам продумывает стратегию тестирования;
- Данный метод разграничивает тестирование разработчиком и тестировщиком.

Недостатками тестирования методом серого ящика являются:

- Возможно лишь частичное покрытие кода.
- В распределённых приложениях трудно связать выявления багов. [18]

1.4.3. Метод белого ящика

Метод тестирования белого ящика заключается в том, что тестировщик имеет неограниченный доступ к коду программы. Не многие способны анализировать чужой код и заниматься написанием тестов не запуская приложение или программу, базируясь на коде. Поэтому специалистов по тестированию методом белого ящика очень мало. Чаще всего это бывшие разработчики ушедшие в тестирование или люди, увлекающиеся программированием.

При данном методе тестирования создаются тестовые сценарии так, чтобы:

- Попасть в каждое ветвление;
- Пройти хоть раз через все вершины;

- Пройти всеми возможными путями;
- Пройти через вновь добавленные участки;
- Пройти через известные проблемные участки.

Преимуществами тестирования методом белого ящика являются:

- Позволяет найти «скрытые» в коде дефекты;
- Позитивные побочные эффекты (например, обучение команды);
- Нахождение проблем производительности;
- Более надежное разбиение на классы эквивалентности;
- Как правило, ускорение цикла нахождение-исправление

Недостатками тестирования методом белого ящика являются:

- Требуется разбираться в чужом коде;
- Очень мало специалистов, способных проводить тестирование методом белого ящика. [18]

Ниже приведена таблица, в которой отражены различия тестирования методами белого и черного ящиков.

Таблица 1.

Отличия методов черного и белого ящиков

Критерии	**Черный ящик**	**Белый ящик**
Основной уровень применимости	Приемочное тестирование	Юнит-тестирование
Ответственный	Независимый тестировщик	Разработчик
Знание программирования	Не обязательно	Необходимо
Знание реализации	Не обязательно	Необходимо
Знание сценариев использования	Необходимо	Необязательно
Основа тестовых сценариев	Спецификации	Код

Таким образом, существует 3 основных метода тестирования, и все они находят свое применение на практике. В компании Schlumberger инженеры по качеству применяют следующие два метода: черного и серого ящиков.

Специалисты по тестирования имеют доступ к функциональности программного обеспечения и их базам данных, если таковые имеются, но не могут видеть код разработанного продукта.

1.5. Виды тестирования по степени автоматизированности

1.5.1. Ручное тестирование

Ручное тестирование представляет собой тестирование программы тестировщиком так, как если бы он был обычным пользователем, а именно без каких-либо дополнительных программ. Чаще всего существует заранее заготовленный план, по которому происходит весть процесс тестирование, чтобы избежать пропусков каких-либо важных пунктов проверки.

Ручное тестирование – это один из важнейших этапов в разработке программного обеспечения. Тестировщик может не соблюдать строго утвержденный план и отклоняться него для более полного тестирования, что позволяет приблизиться к тем ситуациям, когда программа была бы в руках обычного пользователя.

Крупные проекты наоборот, они стараются придерживаются строгой методологии тестирования в целях выявления максимального количества дефектов. Системный подход к тестированию включает в себя несколько этапов:

1) Выбор методов тестирования, установка необходимого оборудования;
2) Написание тестов с подробным описанием их выполнения и ожидаемыми результатами;
3) Выполнение написанных тестов тестировщиками, которые выполняют их вручную и по ходу выполнения записывают результаты;
4) Написание отчета о проведенном тестировании и передача его разработчикам для обсуждения и исправления всех найденных проблем.

Невозможно и неправильно полностью стараться автоматизировать ручное тестирование, так как автоматические тесты могут пропустить множество ошибок. Ручное тестирование более гибкое, чем автоматическое. Используя автоматизированные средства, трудно изменить значение в программе, после начала тестирования. При ручном тестировании, возможно быстро проверить результаты, что позволит увидеть, какой функционал работает лучше всего. [7]

1.5.2. Автматизированное тестирование

Автоматизация тестирования – это процесс проверки программного продукта без участия человека. Автоматизированное тестирование позволяет провести проверку основных функций и шагов автоматически. Чаще всего автоматизации подвергаются те пункты проверки, процесс проверки которых не меняется.

Существует ряд преимуществ автоматизированного тестирования по сравнению с ручным:

1) Повышается качество тестирования ввиду того, что при автоматической проверке «человеческий фактор» не оказывает влияния на качество тестирования;
2) Возможность выполнения таких видов тестирования, которые либо не могут быть выполнены вручную, либо требуют значительных затрат (дополнительное оборудование, персонал);
3) Ускорение процесса тестирования без потери качества.
4) В ходе автоматического тестирования все результаты работы приложения сохраняются автоматически.

Применение средств автоматизации для тестирования программного обеспечения актуально в следующих случаях:

- Для проверки часто использующейся функциональности, которая обладает высоким уровнем рисков на ошибки.
- Для рутинных операций.

- Для тестирования данных, где нужны точные математические расчеты.

В зависимости от предъявляемых заказчиком требований и возможностей инструмента, автоматическому тестированию подлежат и другие программные продукты. Разработка отдельных тест-кейсов автоматизации делает процесс тестирования более эффективным. Создание начальных условий, минимально влияющих на другие тесты, позволяет использовать сценарии, способные вернуть систему к исходному состоянию. [1]

1.5.3. Сравнение ручного и автоматизированного тестирования

Результаты сравнения приведены в Таблице № 1. Данная таблица отражает тенденцию современного тестирования, которое ориентирующуюся на максимальную автоматизацию процесса тестирования и генерацию тестового кода. Но, несмотря на все преимущества автоматизированного тестирования, так же видны и достоинства ручного тестирования. Благодаря тому, что оно является более гибким, то с его помощью можно отследить ошибки, которые очень трудно обнаружить при автоматизированном тестировании. [13]

Таблица 2.

Сравнение ручного и автоматизированного тестирования

Критерии	**Ручное**	**Автоматизированное**
Задание входных значений	Гибкость в задании данных позволяет использовать разные значения на разных циклах прогона тестов, расширяя покрытие	Входные значения строго заданы
Проверка результата	Гибкая, позволяет тестировщику оценивать нечетко сформулированные критерии	Строгая. Нечетко сформулированные критерии

Критерии	Ручное	Автоматизированное
		могут быть проверены только путем сравнения с эталоном
Повторяемость	Низкая. Человеческий фактор и нечеткое определение данных приводят к не повторяемости тестирования	Высокая
Надежность	Низкая. Длительные тестовые циклы приводят к снижению внимания тестировщика	Высокая, не зависит от длины тестового цикла
Чувствительность к незначительным изменениям в продукте	Зависит от детальности описания процедуры. Обычно тестировщик в состоянии выполнить тест, если внешний вид продукта и текст сообщений несколько изменились	Высокая. Незначительные изменения в интерфейсе часто ведут к коррекции эталонов
Скорость выполнения тестового набора	Низкая	Высокая
Возможность генерации тестов	Отсутствует. Низкая скорость выполнения обычно не позволяет исполнить сгенерированный набор тестов	Поддерживается

1.6. Тестирование Web приложений

1.6.1. Структура web-приложений

Веб-приложение – это клиент-серверное приложение, в котором пользователь ведет взаимодействие с сервером при поддержке браузера, где сервером выступает веб-сервер. На стороне веб-сервера располагается основная доля приложения. Именно веб-сервер обрабатывает полученные запросы и создает ответ пользователю в согласовании с бизнес-логикой продукта. Вслед за тем браузер преобразовывает полученный ответ от сервера

в графический интерфейс, который впоследствии может наблюдать пользователь.

Web-приложение содержит следующие элементы (см. рис.1.1.):

1. Клиент. Им выступает браузер, который обрабатывает приобретенный ответ от веб-сервера, дабы у пользователя была возможность увидать графический интерфейс программы на странице браузера. В ответе от веб-сервера приходит информация, реализованная с использованием HTML, CSS, JS и т.п. Благодаря данным технологиям браузеру получается «понять», как нужно «отобразить» все то, что он получил в ответе.

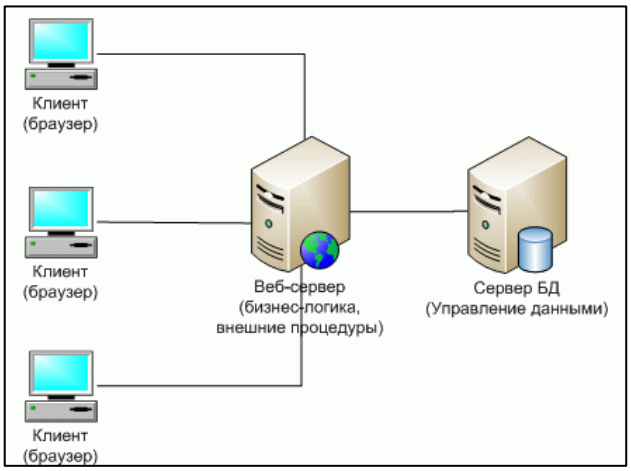


Рис.1.1. Структура web-приложения

2. Веб-сервер. Веб-сервером выступает сервер, который воспринимает HTTP-запросы от пользователей и выдает им HTTP-ответы. Веб-сервер имеет возможность быть любое программное обеспечивание, принимающее запросы, или же компьютер, на котором это программное обеспечивание работает. Самые известные из их это Apache, IIS и NGINX. Тестируемое приложение, чтобы работать на веб-сервере имеет возможность быть реализовано с помощью различных языков программирования : PHP, Python, Ruby, Java, Perl и пр.
3. База данных. Она не считается частью веб-сервера, но нередко приложения не могут выполнять всевозможные на их функции без

нее, например как именно в ней хранится вся динамическая информация программы, например учетные, пользовательские данные и пр. Ввиду такого, что основа данных – это обширное понятие, которое используется во многих сферах, то введем надлежащее определение: основа данных – это информационная модель, позволяющая упорядоченно беречь данные об объекте или же группе объектов, владеющих набором качеств, которые возможно разбить на категории. Она функционирует под управлением систем управления базами данных (далее – СУБД). Наиболее известными СУБД считаются MySQL, MS SQL Server и Oracle.

1.6.2. Особенности тестирования web-приложений

Практически все современные программы нацелены на работу с сетью. Хранение данных web-приложений выполняется, большей частью, на сервере, а обмен информацией случается по сети. Когда наблюдается сбой в сетевой среде, то зачастую непросто понять, где именно это произошло, и вследствие этого режим работы, или же сообщение об ошибке, может стать результатом ошибок, случившихся в различных частях сетевой системы.

Имея большое количество совместного с испытанием классических приложений, испытание web-ориентированных приложений содержит собственные особенности, связанные до этого толко со средой функционирования. Имея компонентные, структурные и технологические особенности, web-приложениям присущи особенности режимов работы, установки, пуска, остановки и удаления, а еще формирования интерфейсов. Ниже приведена таблица, показывающая отличия web-приложений от классических приложений. [19]

Таблица 3.

Сравнение тестирования web и классического приложений.

Отличие	**Web-приложение**	**Классическое приложение**
Технологическое	Работает с использованием принципиально различных технологий	Работает с использованием одной или семейства родственных технологий
Структурное	Приложение — “многокомпонентное”. Состоит из большого числа модулей. Обязательно использует серверы БД, web-серверы, серверы приложений	Приложение – “монолитное”. Состоит из одного или небольшого количества модулей
Режимы работы	Работает в режиме “запрос-ответ”, т.е. известно о некотором наборе действий только после запроса на сервер	Работает в режиме реального времени, т.е. известно о действиях пользователя сразу же, как только оно выполнено
Формирования интерфейса	Использует для формирования пользовательского интерфейса стремительно развивающиеся технологии, множество которых конкурирует между собой	Использует для формирования интерфейса пользователя относительно устоявшиеся и стандартизированные технологии
Работа с сетью	Активно использует сетевые каналы передачи данных	Практически не использует сетевые каналы передачи данных
Запуск и остановка	Запускается и останавливается по факту поступления каждого запроса, т.е. очень часто	Запускается и останавливается редко
Количество пользователей	Количество пользователей, одновременно использующих приложение, сложно прогнозируемо и может скачкообразно меняться в широких диапазонах	Количество пользователей, одновременно использующих приложение, подвержено контролю, ограничено и легко прогнозируемо

Отличие	Web-приложение	Классическое приложение
Инсталляция и деинсталляция	Процесс инсталляции часто недоступен конечному пользователю. Инсталляция требует специфических знаний. Процесс изменения компонент приложения не предусматривается или требует квалификации пользователей. инсталлятор отсутствует.	Процесс стандартизирован и максимально ориентирован на широкую аудиторию пользователей

1.7. Инструмент Selenium, как средство для автоматизации тестирования WEB-приложений

В сфере тестирования программного обеспечения инструменты автоматизации, безусловно, играют важную роль. Существует огромное количество инструментов, которые могут справиться с этой задачей: UFT, Watir, TestComplite и другие. Но самым популярным фреймворком, предназначенным для автоматизации тестирования веб-приложений является Selemium. Именно поэтому он и был выбран компанией Schlumberger для написания автоматических тестов для тестирования порталов.

Selenium – это проект, в рамках которого разрабатывается серия программных продуктов с открытым исходным кодом: Selenium WebDriver, Selenium RC, Selenium Server, Selenium Grid, Selenium IDE. [6] Рассмотрим назначение этих компонентов.

1.7.1. Selenium WebDriver

Selenium WebDriver – это программная библиотека для управления браузерами и является основным продуктом, разрабатываемым в рамках проекта Selenium.

WebDriver представляет собой семейство драйверов для различных браузеров и набор клиентских библиотек для этих драйверов на разных языках программирования (см. Рис. 1.2.):

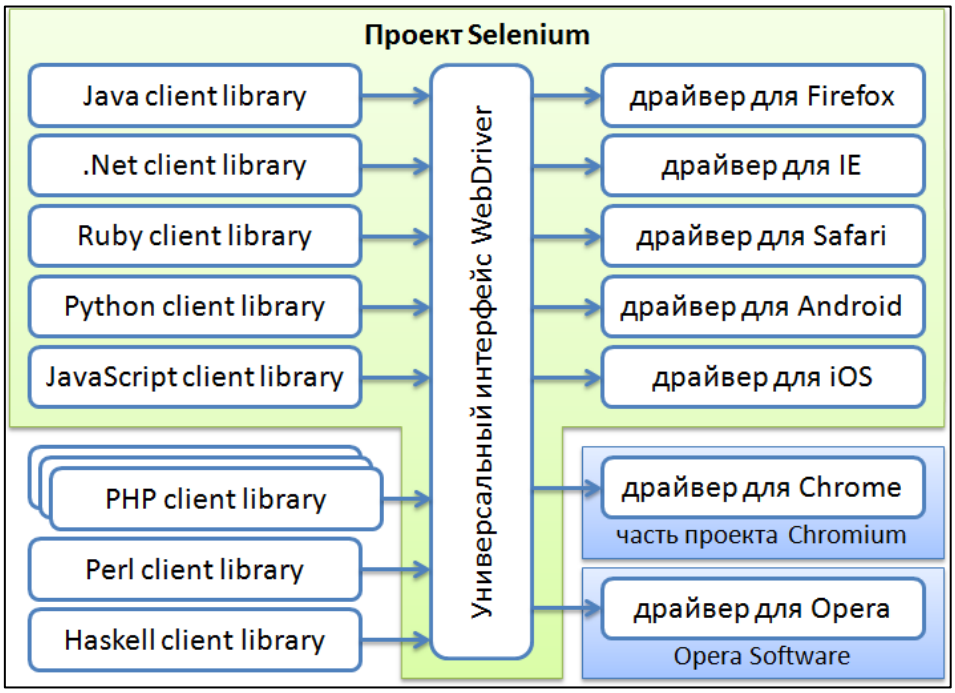


Рис.1.2. Проект Selenium

В рамках проекта Selenium разрабатываются драйвера для следующих браузеров: Firefox, Internet Explorer и Safari. Также разрабатываются драйвера и для мобильных браузеров Android и iOS.

Подобная обстановка и с клиентскими библиотеками – в рамках проекта Selenium разрабатываются библиотеки для языков Java, .Net (C#), Python, Ruby и JavaScript. [6]

1.7.2. Selenium RC

Selenium RC – это предшествующая версия библиотеки для управления браузерами и является средством для «удалённого» управления браузером.

Данная версия с функциональной точки зрения значительно уступает WebDriver. В данный момент она располагается в замороженном состоянии, не развивается и в том числе и не исправляются известные баги. А всем, кто встречается с лимитированиями Selenium RC, предлагается начать внедрение WebDriver. [6]

1.7.3. Selenium Server

Selenium Server – это сервер, который позволяет управлять браузером с удалённой машины, по сети. Сначала на той машине, где должен работать браузер, устанавливается и запускается сервер. Затем на другой машине запускается программа, которая, используя специальный драйвер RemoteWebDriver, соединяется с сервером и отправляет ему команды. Он в свою очередь запускает браузер и выполняет в нём эти команды, используя драйвер, соответствующий этому браузеру (см. рис 1.3):

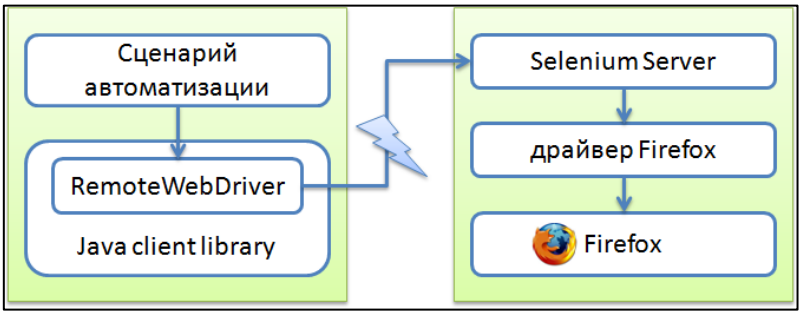


Рис.1.3. Selenium Server поддерживает одновременно два набора команд – для новой версии (WebDriver) и для старой версии (Selenium RC).

1.7.4. Selenium Grid

Selenium Grid – это кластер, состоящий из нескольких Selenium-серверов. Он предназначен для организации распределённой сети, позволяющей параллельно запускать много браузеров на большом количестве машин.

Selenium Grid имеет топологию «звезда». Сеть может быть гетерогенной , то есть коммутатор и узлы могут работать под управлением разных операционных систем, на них могут быть установлены разные браузеры. Одна из задач Selenium Grid заключается в том, чтобы «подбирать» подходящий узел, когда во время старта браузера указываются требования к нему – тип браузера, версия, операционная система, архитектура процессора и ряд других атрибутов. [6]

1.7.5. Selenium IDE

Selenium IDE – плагин к браузеру Firefox, который может записывать действия пользователя, воспроизводить их, а также генерировать код для WebDriver или Selenium RC, в котором выполняются те же самые действия. [6]

1.8. Вывод по главе: общие вопросы организации тестирования программного обеспечения

Тестирование программного обеспечения включает в себя сочетание различных видов (функциональное, стрессовое, нагрузочное, тестирование производительности, стабильности, удобства использования, безопасности, совместимости) и методов (черного, белового и серого ящиков) тестирования, а также тестирование может быть ручным, автоматизированным или же включать в себя сразу оба подхода. Все это позволяет добиться наивысшего качества выпускаемого продукта. Несмотря на то, что между классическим приложением и web-приложением, есть ряд отличий, для тестирования применяются одни и те же методы и подходы. Для web-приложений есть ряд инструментов для автоматизации тестирования. Компанией Schlumberger был выбран инструмент для автоматизированного тестирования Selenium WebDriver ввиду того, что он прост в понимании, широко используется, а также позволяет работать со всеми необходимыми для проверки наших приложений браузерами (Fierfox, Google Chrome и Internet Explorer) и писать тесты на 6 языках программирования (C#, Python, Java, Ruby, PHP и Java Script). Автоматизация позволяет сократить время на тестирование и избежать пропуска багов ввиду человеческого фактора.

Глава 2. Разработка алгоритмов для нахождения тестовых наборов

При разработке программного обеспечения разрабатываются тестовые сценарии. Одна из главных задач инженера по качеству – это планирование тестов, а для этого необходимо выбрать именно тот подход, который поможет обеспечить максимальное качество за отведенное время на тестирования с оптимальным количеством тестов. Для этого существует множество подходов. Для тестирования продуктов в компании Schlumberger были выбраны следующие подходы:

1) Полный перебор
2) Минимальный перебор
3) Атомарный перебор
4) Попарные проверки [3]

Каждый из подходов имеет свои достоинства и недостатки, поэтому для каждой задачи выбор подхода индивидуален. Ввиду того, что вышеуказанные методы описываются и объясняются в большей части на разных форумах в интернете и вебинарах, то формального их описания и четко задокументированного алгоритма нет. Необходимо было вывести алгоритмы [21] и описать их на математическом языке. Но для начала, введем следующие определения:

Фактор – набор параметров (условий) содержащий конкретные значения для проверки.

Переменная – одно из значений фактора.

Тестовый сценарий – совокупность переменных, каждая из который представляет некоторое значение соответствующего фактора.

Тестовый набор – совокупность всех тестовых сценариев, сгенерированных с использованием одного из подходов.

Рассмотрим далее описание различных методов (подходов) построения тестовых наборов и дадим формальное представление алгоритмов реализации каждого из них.

2.1. Постановка задачи

Имеем набор из N тестовых независимых факторов: f_1, f_2, …, f_N, где каждый фактор f_i имеет L_i возможных переменных: $f_i = \{l_{i,1}, l_{i,2}, \ldots, l_{i,Li}\}$.

Необходимо создать тестовый набор T.

Введем следующие обозначения, которые будут использоваться в описании подходов и алгоритмах:

step – количество подряд идущих параметров с одним значением;

C – количество тестовых наборов;

L – набор всех факторов и переменных;

i – порядковый номер фактора;

j – порядковый номер значения в факторе;

L[i].Count – количество параметров в i факторе;

L[j][k] – значение параметра в факторе j;

T – тестовый набор;

T[i] – один тестовый случай;

T[i][j] – значение тестового набора;

V – количество параметров, которые уже были использованы в одном из тестов.

2.2. Полный перебор

Полный перебор — метод решения математических задач. Относится к классу методов поиска решения исчерпыванием всевозможных вариантов. Сложность полного перебора зависит от количества всех возможных решений задачи.

В рамках данного подхода задача создания тестового набора формулируется в следующем виде:

Необходимо создать набор тестовых сценариев T, включающий всевозможные комбинации значений факторов.

Решение: набор тестовых сценариев T в полном переборе содержит всевозможные комбинации факторных переменных, в связи с этим выведем формулу для нахождения количества тестовых наборов.

Для каждого факторы имеется l_i переменных, где i – номер факторов, к которому он относится. Каждый фактор имеет свой набор переменных, а также их количество. Количество переменных для каждого фактора может находиться в следующих пределах: [2, +∞).

Таким образом имеем для f_1 фактора l_1 переменных, для f_2 фактора l_2 переменных, для f_N фактора имеем l_N переменных. Чтобы узнать количество наборов, необходимо перемножить количество факторов на количество их переменных. Имеем следующую формулу:

$$C = \prod \binom{N}{l_N} = \prod_{i=1}^{N} l_N$$

Набор тестовых сценариев T состоит из всевозможных комбинаций факторных переменных, а также каждый тестовый набор имеет столько параметров, сколько всего факторов. Таким образом имеем C тестовых наборов с N параметрами в каждом: $T[j] = \{f_1, f_2, \ldots, f_N\}$, где T[j] – один тестовый набор, N – количество факторов; $f_1, f_2, \ldots, f_N$ – какой-либо параметр из фактора. Каждый фактор f_i имеет L_i возможных переменных: $f_i = \{l_{i,1}, l_{i,2}, \ldots, l_{i,Li}\}$. Таким образом, имеем следующий набор тестовых сценариев:

$$T = \begin{Bmatrix} l_{1,1}, & l_{2,1}, & . & . & . & , & l_{N-1,1}, & l_{N,1} \\ l_{1,1}, & l_{2,1}, & . & . & . & , & l_{N-1,1}, & l_{N,2} \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ l_{1,l_1}, & l_{r,l_2}, & \ldots, & & & & l_{N-1,l_{N-1}}, & l_{N,l_N-1} \\ l_{1,l_1}, & l_{r,l_2}, & \ldots, & & & & l_{N-1,l_{N-1}}, & l_{N,l_N} \end{Bmatrix}$$

Блок схема алгоритма нахождения тестовых наборов методом полного перебора показана на рисунке 2.1.

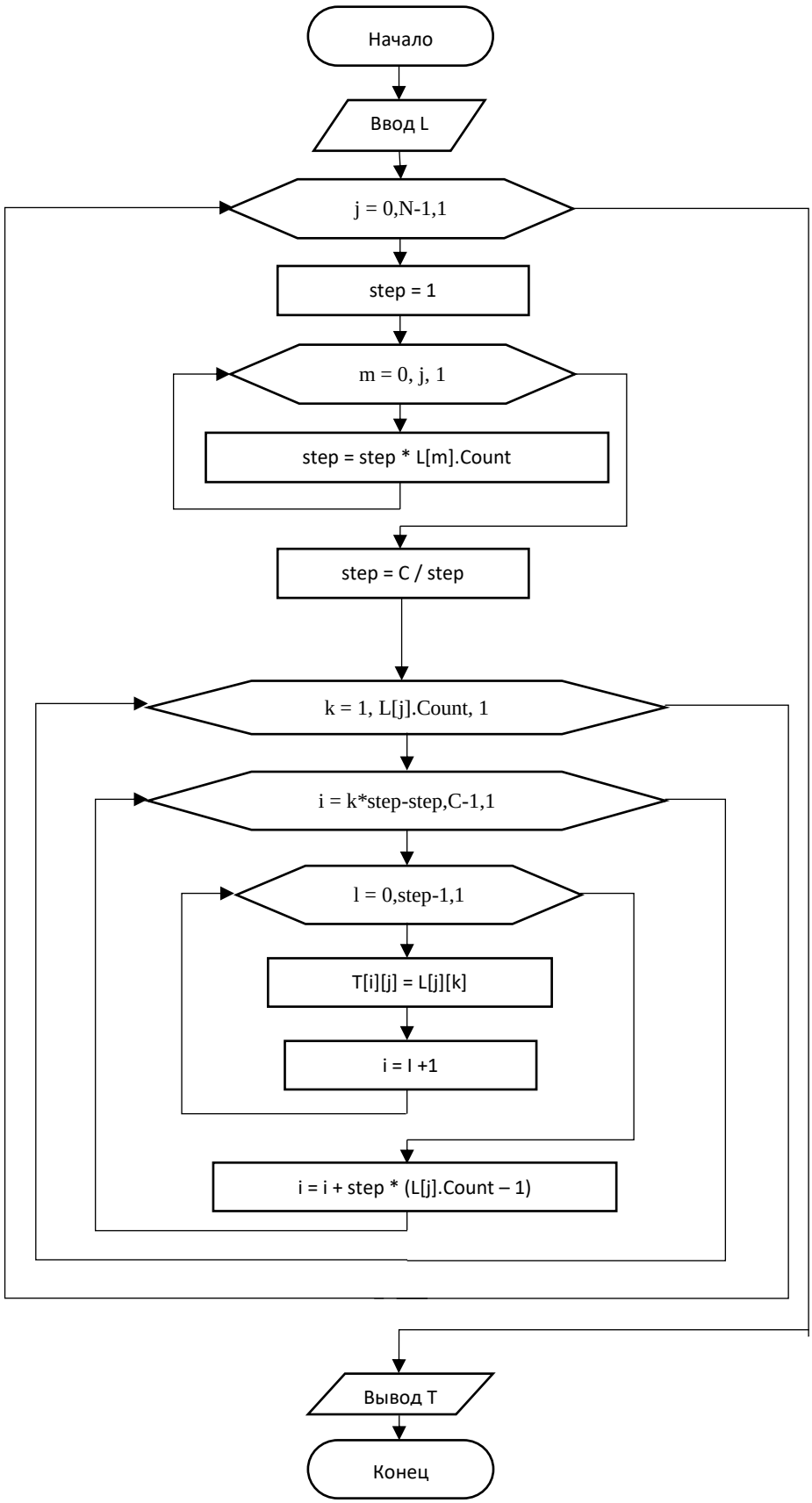


Рис. 2.1. Блок-схема алгоритма полного перебора

2.3. Способ минимальных проверок

Метод минимальных проверок включает в себя минимальное количество необходимых тестов. Для нахождения тестового набора выбирается параметр, в котором больше всего значений переменных – это и будет количество тестов методом минимальных проверок. Все остальные параметры, когда все значения уже были использованы в каком-либо тестовом сценарии, используется тот, который наиболее часто используется для решения данной задачи.

Необходимо создать набор тестовых сценариев T, состоящий из минимально возможного количества тестов с использованием всех имеющихся значений.

Решение: Набор тестовых сценариев T, созданный с использованием метода минимальных проверок, содержит ровно столько тестовых сценариев, сколько параметров в факторе с наибольшим количеством значений.

Для каждого факторы имеется li переменных, где i – номер фактора, к которому он относится. Каждый фактор имеет свой набор переменных, а также их количество. Количество переменных для каждого фактора может находиться в следующих пределах: [2, +∞).

Таким образом, чтобы узнать количество наборов, необходимо найти фактор с наибольшим количеством переменных:

$$C = \max_{0 \le i \le 1}(l_i)$$

Тестовый набор T состоит из C тестовых сценариев, а каждый сценарий содержит столько параметров, сколько всего факторов. Таким образом имеем C тестовых наборов с N параметрами в каждом: $T[j] = \{f_1, f_2, \ldots, f_N\}$, где T[j] – один тестовый набор, N – количество факторов; f_1 , f_2, … , f_N – какой-либо параметр из фактора. Каждый фактор f_i имеет L_i возможных переменных: $f_i = \{l_{i,1}, l_{i,2}, \ldots, l_{i,Li}\}$.

Пусть максимальное количество переменных будет в факторе $f_{\frac{N}{2}}$, а значения в факторе расположены в порядке от самого часто используемого до

наименее вероятного. Таким образом, имеем следующий набор тестовых сценариев:

$$T = \begin{Bmatrix} l_{1,1},\ l_{2,1}, \dots, l_{\frac{N}{2},1}, \dots,\ l_{N-1,1},\ l_{N,1} \\ l_{1,2},\ l_{2,2}, \dots, l_{\frac{N}{2},2}, \dots,\ l_{N-1,2},\ l_{N,2} \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\ l_{1,1},\ l_{2,1}, \dots, l_{\frac{N}{2},l_{\frac{N}{2}}}, \dots,\ l_{N-1,1},\ l_{N,1} \end{Bmatrix}$$

Блок схема алгоритма нахождения тестовых наборов методом полного перебора показана на рисунке 2.2.

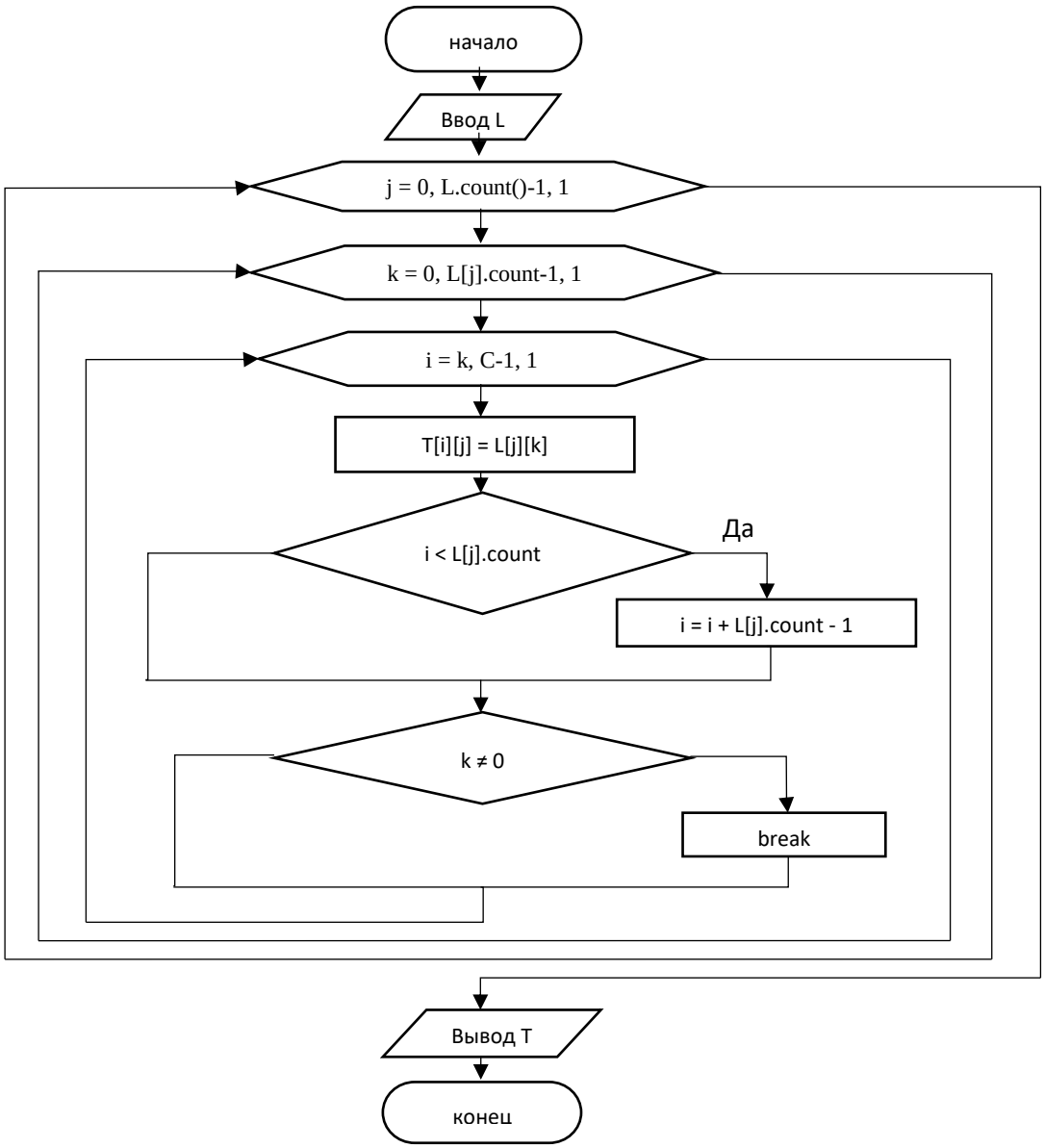


Рис. 2.2. Блок схема алгоритма для нахождения тестовых наборов методом минимальных проверок

2.4. Атомарные проверки

Метод атомарных проверок – это метод, в котором с каждым последующим тестовым случаем изменяется лишь одно значения для одного из параметров относительно тех, что установлены по умолчанию. Метод заключается в следующем: для каждого параметра выбирается лишь одно значение, которое определяется, как наиболее вероятное, а далее в каждом последующем тесте заменяется значение лишь для одного из параметров, пока не будут перебраны всевозможные значения для каждого из параметров.

Необходимо создать набор тестовых сценариев T методом атомарных проверок.

Решение: Набор тестовых сценариев T в методе атомарных проверок состоит из сценариев, где в каждом последующем тесте заменяется лишь одно значение для одного фактора относительно первого тестового набора. Выведем формулу для нахождения количества тестовых наборов.

Для каждого факторы имеется l_i переменных, где i – номер факторов, к которому он относится. Каждый фактор имеет свой набор переменных, а также их количество. Количество переменных для каждого фактора может находиться в следующих пределах: [2, +∞) (на практике обычно не превышает 6).

Таким образом, имеем для f_1 фактора l_1 переменных, для f_2 фактора l_2 переменных, для f_N фактора имеем l_N переменных. В первом тесте будет задействовано по одному значению из каждого фактора, и этот набор будет взят за основу для всех последующих, поэтому он будет включать в себя значения, которые на практике будут применяться вероятнее всего. Во втором тесте будет заменено значение лишь для первого фактора. Тестов, где заменяется значение первого фактора будет l_1-1, для второго l_2-1, для l_n l_n-1. Таким образом тестовых сценариев, если исключить первый, будет равно:

$$\sum_{i=0}^{N}(l_i - 1) = \sum_{i=0}^{N}(l_i) - N \ .$$

Добавим первый тест и получим следующую формулу:

$$C = \sum_{i=0}^{N}(l_i) - N + 1$$

Набор тестовых сценариев T состоит из всевозможных комбинаций факторных переменных, а также каждый тестовый набор имеет столько параметров, сколько всего факторов. Таким образом, имеем r тестовых наборов с N параметрами в каждом: T[j] = {f_1, f_2, …, f_N}, где R[j] – один тестовый набор, N – количество факторов; f_1 , f_2, … , f_N – какой-либо параметр из фактора. Каждый фактор f_i имеет l_i возможных переменных: f_i = {$l_{i,1}$, $l_{i,2}$, …, $l_{i,Li}$}. Пусть в каждом факторе переменные расположены в порядке от самого часто используемого, до менее используемого. Так как в каждом последующем тесте заменяется лишь одно значение одного фактора относительно первого тестового сценария, то имеем следующий набор тестовых сценариев:

$$T = \begin{Bmatrix} l_{1,1},\ l_{2,1},\ \dots\ ,\ l_{N-1,1},\ l_{N,1} \\ l_{1,2},\ l_{2,1},\ \dots\ ,\ l_{N-1,1},\ l_{N,1} \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\ l_{1,1},\ l_{r,1},\ \dots\ ,\ l_{N-1,1},\ l_{N,l_N-1} \\ l_{1,1},\ l_{r,1},\ \dots\ ,\ l_{N-1,1},\ l_{N,l_N} \end{Bmatrix}$$

Блок схема алгоритма для нахождения количества тестовых наборов для метода атомарных проверок показан на рисунке 2.3.

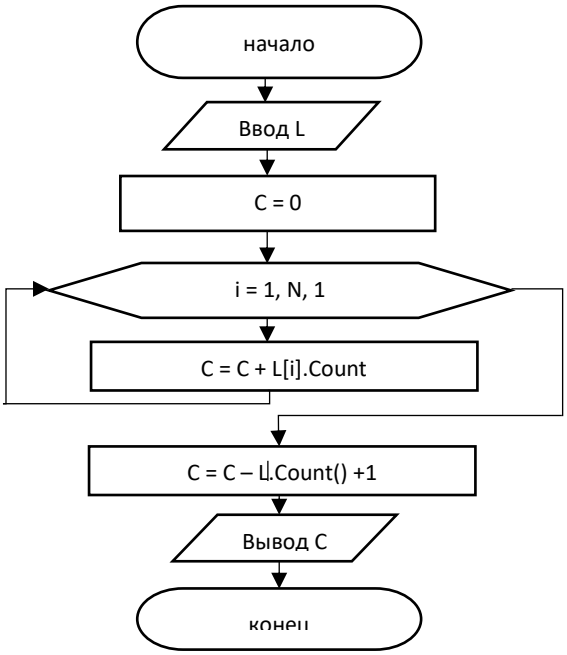


Рис. 2.3. Блок-схема алгоритма нахождения количества тестовых наборов для метода атомарных проверок

Алгоритм нахождения тестовых наборов с помощью метода атомарных проверок показан на рисунке 2.4.

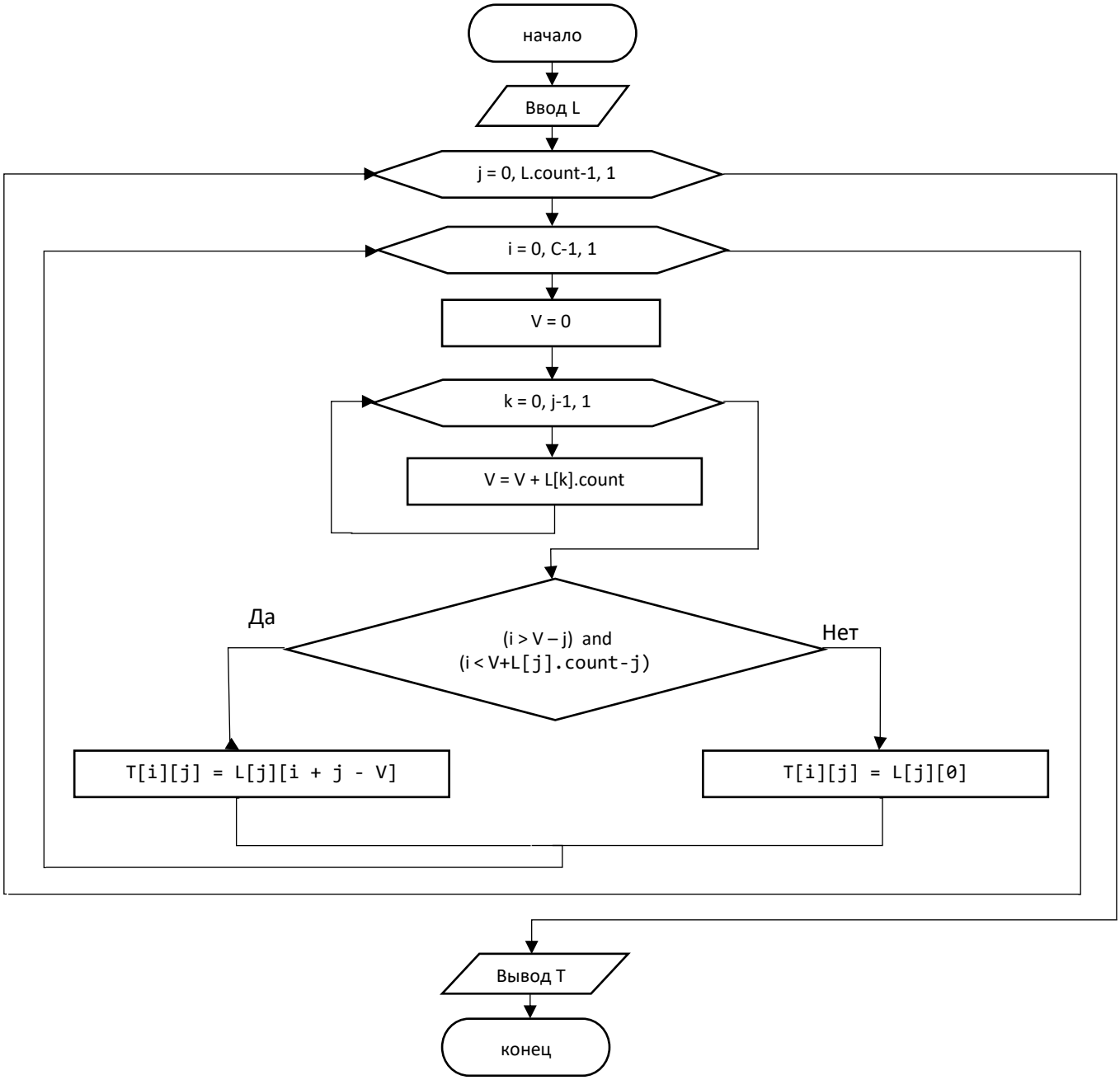


Рис. 2.4. Блок схема алгоритма атомарных проверок

2.5. Попарные проверки

Попарное тестирование – это комбинаторный метод тестирования программного обеспечения, который для каждой пары входных параметров системы проверяет всевозможные дискретные комбинации этих параметров. [2]

Необходимо создать набор тестовых сценариев T, где задействованы все пары факторных значений.

Решение:

Набор тестовых сценариев R в методе попарных проверок состоит и сценариев, где содержатся все комбинации пар переменных.

Для каждого факторы имеется l_i переменных, где i – номер факторов, к которому он относится. Каждый фактор имеет свой набор переменных, а также их количество. Количество переменных для каждого фактора может находиться в следующих пределах: [2, +∞).

Таким образом, имеем для f_1 фактора l_1 переменных, для f_2 фактора l_2 переменных, для f_N фактора имеем l_N переменных. Как правило, количество тестовых наборов для метода попарных проверок — это произведение количества для двух факторов с максимальным числом параметров.

Таким образом, тестовых сценариев будет равно:

$$r = \max_{0 \le i \le 1}(l_i) * \max_{0 \le i \le 1}(l_i \setminus \max_{0 \le i \le 1}(l_i))$$

где r – количество тестовых наборов.

Набор тестовых сценариев R состоит из всевозможных комбинаций пар факторных переменных, а также каждый тестовый набор имеет столько параметров, сколько всего факторов. Таким образом имеем r тестовых наборов с N параметрами в каждом: $R[j] = \{f_1, f_2, \dots, f_N\}$, где R[j] – один тестовый набор, N – количество факторов; f_1 , f_2, … , f_N – какой-либо параметр из фактора. Каждый фактор f_i имеет l_i возможных переменных: $f_i = \{l_{i,1}, l_{i,2}, \dots, l_{i,Li}\}$. Все факторы отсортированы по количеству переменных в них. Следовательно, фактор l_1 имеет самое большое количество переменных, l_2 –

второе по величине, либо равное количество переменных относительно первого, а l_N минимальное. Так как необходимо перебрать всевозможные комбинации пар, а в одном тестовом наборе могут сочетаться сразу несколько пар, то для первого тестового набора из N факторов имеем следующие пары: $\{l_{1,1}, l_{2,1}\}$, $\{l_{1,1}, l_{3,1}\}$, …, $\{l_{1,1}, l_{N,1}\}$. Таким образом, имеем следующий набор тестовых параметров.

Если $l_2 \neq l_3$:

$$T = \left\{ \begin{array}{l} l_2 \left\{ \begin{array}{l} l_{1,1}\,,\ l_{2,1}\,, l_{3,1}\,, \dots, l_{N,1} \\ l_{1,1}\,,\ l_{2,2}\,, l_{3,2}\,, \dots, l_{N,2} \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots \\ l_{1,1}\,, \dots\,, l_{3,l_3}\,, \dots, \dots \\ l_{1,1}\,, \dots\,, l_{3,1}\,, \dots, \dots \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots \\ l_{1,1}\,,\ l_{2,l_2}\,, \dots\,, \dots, \dots \end{array} \right. \\ l_2 \left\{ \begin{array}{l} l_{1,2}\,,\ l_{2,1}\,, \dots\,, \dots, \dots \\ l_{1,2}\,,\ l_{2,2}\,, \dots\,, \dots, \dots \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots \\ l_{1,2}\,,\ l_{2,l_2}\,, \dots\,, \dots, \dots \end{array} \right. \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\ l_2 \left\{ \begin{array}{l} l_{1,l_1}\,,\ l_{2,1}\,, \dots\,, \dots, \dots \\ l_{1,l_1}\,,\ l_{2,2}\,, \dots\,, \dots, \dots \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots \\ l_{1,l_1}\,,\ l_{2,l_2}\,, \dots\,, \dots, \dots \end{array} \right. \end{array} \right\}$$

Если $l_2 = l_3$:

$$T = \left\{ \begin{array}{l} l_2 \left\{ \begin{array}{l} l_{1,1}\,,\ l_{2,1}\,, l_{3,1}\,, \dots, l_{N,1} \\ l_{1,1}\,,\ l_{2,2}\,, l_{3,2}\,, \dots, l_{N,2} \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots \\ l_{1,1}\,,\ l_{2,l_2}\,,\ l_{3,l_3}\,, \dots, \dots \end{array} \right. \\ l_2 \left\{ \begin{array}{l} l_{1,1}\,,\ l_{2,1}\,, l_{3,2}\,, \dots, \dots \\ l_{1,1}\,,\ l_{2,2}\,, l_{3,3}\,, \dots, \dots \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots \\ l_{1,1}\,,\ l_{2,l_2-1}\,, l_{3,l_3}\,, \dots, \dots \\ l_{1,1}\,,\ l_{2,l_2}\,, l_{3,1}\,, \dots, \dots \end{array} \right. \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\ l_2 \left\{ \begin{array}{l} l_{1,l_1}\,,\ l_{2,1}\,, \dots\,, \dots, \dots \\ l_{1,l_1}\,,\ l_{2,2}\,, \dots\,, \dots, \dots \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots \\ l_{1,l_1}\,,\ l_{2,l_2}\,, \dots\,, \dots, \dots \end{array} \right. \end{array} \right\}$$

Блок схема алгоритма для нахождение количества тестовых наборов для метода попарных проверок представлен на рисунке 2.5, а блок схема алгоритма для нахождение тестовых наборов для метода попарных проверок представлен на рисунке 2.6.

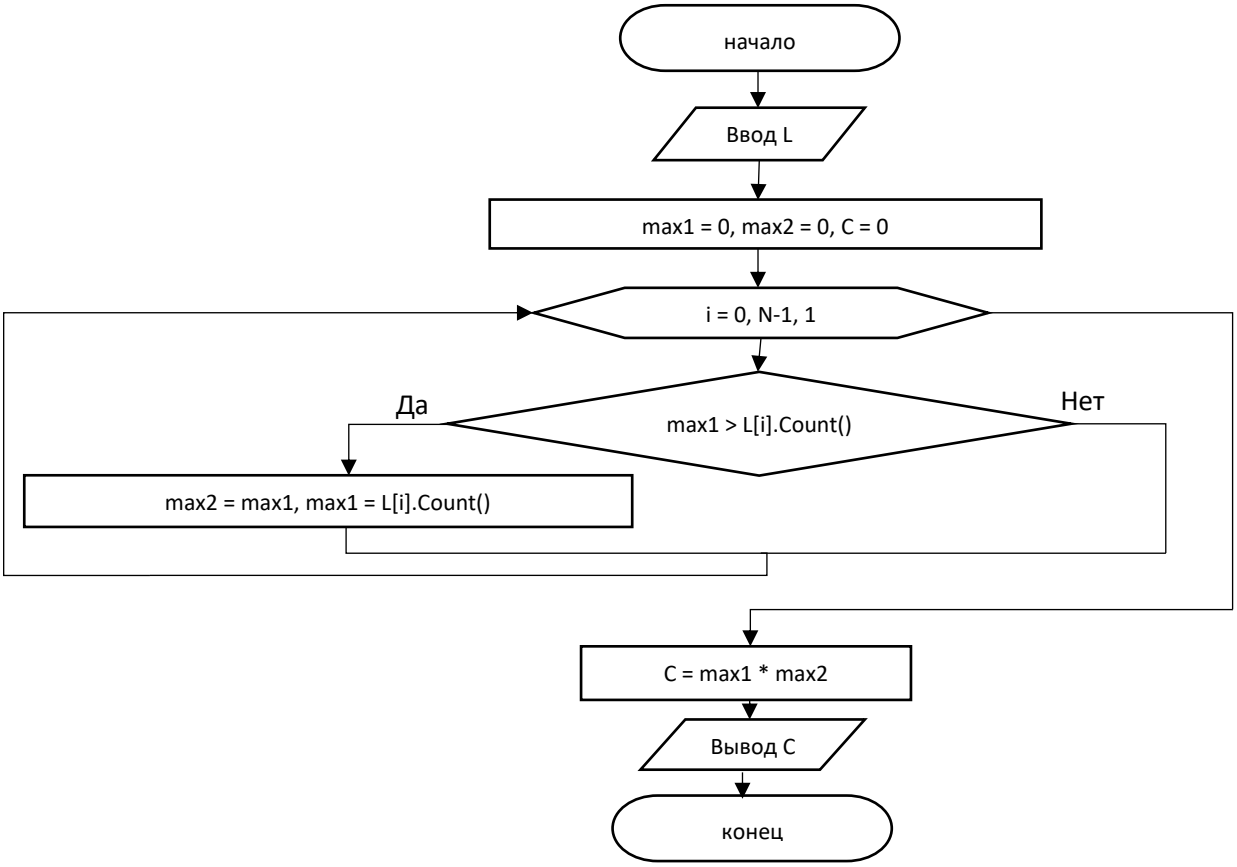
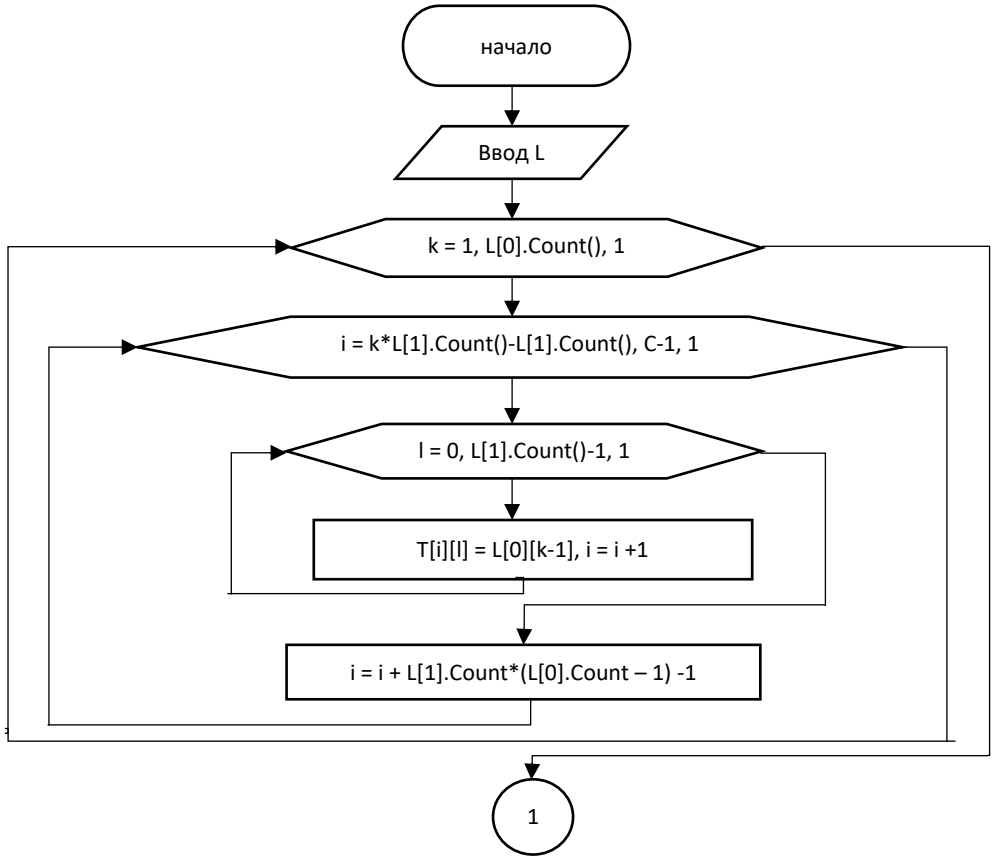


Рис. 2.5. Блок-схема алгоритма нахождения количества тестовых наборов для метода попарных проверок



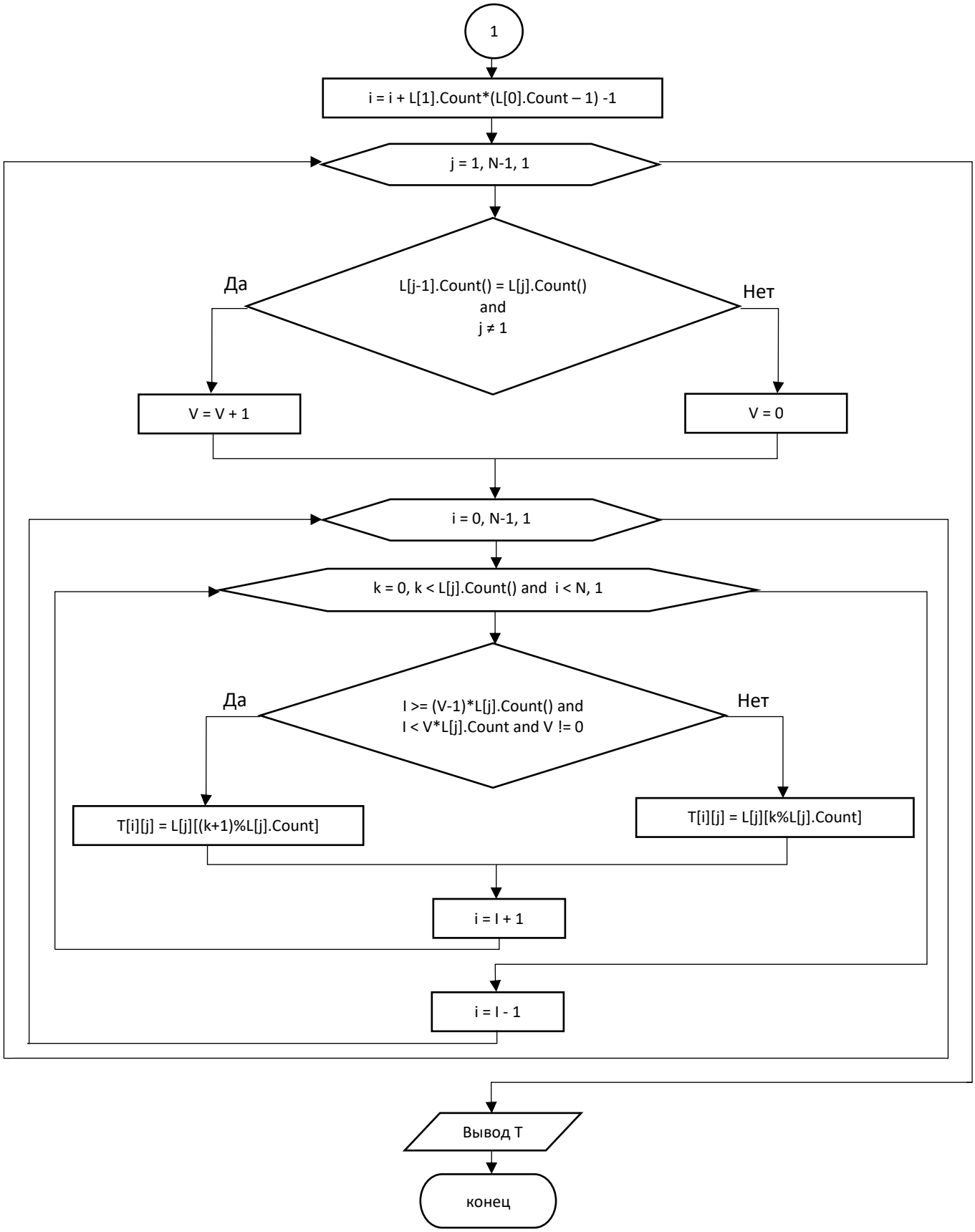


Рис. 2.6. Блок-схема алгоритма нахождения тестовых наборов методом попарных проверок

2.6. Оценка сложности алгоритмов

При разработке алгоритмов очень важно иметь возможность оценить ресурсы, необходимые для проведения испытаний. Результатом оценки является функция вычислительной сложности. Оценка позволяет предсказать время выполнения и сравнивать эффективность алгоритмов. Проведем оценку сложности вышеописанных алгоритмов, но для начала введем следующие обозначения: пусть n – количество факторов, m – максимальное число факторных переменных среди всех факторов. Оценку сложности алгоритмов будем проводить для наихудшего случая, когда количество факторных переменных для каждого фактора одинаково. [9, 14]

2.6.1. Оценка сложности алгоритма полного перебора

Алгоритм содержит четыре уровня вложенных циклов. Проведем оценку сложности каждого из них. Самый первый цикл представляет собой линейную скорость поскольку цикл выполняет неизменное число команд. Следующий цикл находится внутри первого и представляет собой логарифмическую скорость, так как количество операций будет уменьшаться в зависимости от уменьшения N (количества факторов). Итого сложность первых двух циклов O(n)* O(n/2).

Третий цикл является вложенным в первый и максимальное число раз будет проходить m раз. Алгоритм имеет линейную скорость, которая равна O(m). Так как этот алгоритм является вложенным в первый, то их общая скорость будет равна O(n)*O(m). В третий цикл вложено еще два. Первый из них напрямую зависит от количества переменных в факторе и будет выполнятся максимальное число раз для фактора с максимальным числом переменных. Таким образом, скорость его линейна и равна O(m). Второй вложенный цикл проходит также напрямую зависит от количества переменных в факторах и максимальное число выполнений операций в нем будет равняться m. Ввиду этого имеем сложность алгоритма O(m).

Посчитаем сложность всего алгоритма: $O(n)* O(n/2) + O(n)* O(m)* O(m)* O(m) = O(n^2/2)+ O(nm^3)$. Так как n и m могут быть в переделах $[2, +\infty)$, то функция m^3 будет возрастать быстрее n^2. Таким образом, сложность алгоритма полного перебора равна $O(nm^3)$.

2.6.2. Оценка сложности алгоритма минимальных проверок

Алгоритм содержит три уровня вложенных циклов. Проведем оценку сложности каждого из них. Самый первый цикл представляет собой линейную скорость поскольку цикл выполняет неизменное число команд в зависимости от количества фактором, а именно n раз.

Следующий цикл находится внутри первого и представляет собой так же линейную скорость, так как выполняется столько, сколько переменных содержится в факторе. Таким образом, сложность алгоритма составляет O(m).

Третий вложенный цикл зависит от количества тестов, а их число равно максимальному числу переменных среди всех факторов, то есть m. Таким образом сложность третьего цикла равно O(m).

Посчитаем сложность всего алгоритма: $O(n)*O(m)*O(m) = O(nm^2)$. Таким образом, сложность алгоритма минимальных проверок равна $O(nm^2)$.

2.6.3. Оценка сложности алгоритма атомарных проверок

Алгоритм содержит три уровня вложенных циклов. Проведем оценку сложности каждого из них. Самый первый цикл представляет собой линейную скорость поскольку цикл выполняет неизменное число команд в зависимости от количества фактором, а именно n раз.

Следующий цикл находится внутри первого и представляет собой так же линейную скорость, но выполняется столько раз, сколько всего тестов. Количество факторных переменных равно m. Количество факторов равно n. Таким образом, сложность алгоритма составляет $O(nm – n + 1) = O(nm)$.

Третий вложенный цикл представляет собой линейную скорость, так как будет проходить ровно N (количества факторов) раз, поэтому, сложность третьего цикла равна O(n).

Посчитаем сложность всего алгоритма: $O(n) * O(nm) * O(n) = O(n^3m)$. Таким образом, сложность алгоритма атомарных проверок равна $O(n^3m)$

2.6.4. Оценка сложности алгоритма попарных проверок

Алгоритм содержит два трехуровневых цикла. Проведем оценку сложности каждого из них. Самый первый цикл представляет собой линейную скорость поскольку цикл выполняет неизменное число команд в зависимости от количества переменных в первом факторе (при данном подходе в первом факторе содержится максимальное число переменных среди всех факторов), а именно m раз.

Следующий цикл находится внутри первого и представляет собой так же линейную скорость, но выполняется столько раз, сколько всего переменных во втором факторе. Их число может быть равно первому фактору, а значит максимальному. Таким образом, скорость алгоритма равна $O(m)$. Третий цикл имеет линейную скорость и зависит так же, как и второй от количества переменных во втором факторе. Следовательно, его скорость также равна $O(m)$.

Таким образом, сложность первого трехуровневого цикла равна $O(m) * O(m) * O(m) = O(m^3)$.

Проведем оценку второго трехуровневого цикла. Самый первый цикл представляет собой линейную скорость поскольку цикл выполняет неизменное число команд в зависимости от количества фактором, а именно n раз.

Первый вложенный в него цикл имеет также линейную скорость и зависит от количества факторов. Следовательно, скорость цикла равна $O(n)$.

Второй вложенный цикл зависит от количества переменных в факторе. Так как в самый худший случай – это когда количество всех факторных переменных равно, то скорость алгоритма равна $O(m)$.

Таким образом, сложность второго трехуровневого цикла равна $O(n) * O(n) * O(m) = O(n^2m)$.

Вычислим сложность всего алгоритма: $O(m^3) + O(n^2*m)$. Таким образом, скорость алгоритма будет равна $O(n^2*m)$.

2.7. Сравнение тестовых подходов

В процессе изучения четырех подходов и применения их на практике, можно сформировать следующую таблицу, которая отражает все плюсы и минусы каждого из подходов.

Таблица 4.

Сравнение тестовых подходов для генерации тестовых наборов

	Полный перебор	Минимальные проверки	Атомарные проверки	Попарные проверки
Глубина покрытия	Полное покрытие	Плохое	Среднее	Хорошее
Сложность алгоритма	$O(nm^3)$	$O(nm^2)$	$O(n^3m)$	$O(n^2m)$
Простота создания	Легко	Легко	Легко	Средне
Локализация дефектов	Легко	Сложно	Легко	Средне

2.7. Вывод по главе: разработка алгоритмов генерации тестовых наборов

В ходе изучения вышеописанных подходов, было замечено, что нигде нет формального описания решения этой задачи и описанных алгоритмов. Таким образом, в данной главе были досконально исследованы эти подходы и формально описаны, были разработаны алгоритмы, с помощью которых можно сгенерировать тестовый набор согласно выбранному подходу, а также оценена сложность каждого из созданных алгоритмов и проведено их сравнение.

Глава 3. Автоматизация тестирования web-приложений

Для автоматизации тестирования web-приложений была выбрана среда разработки Visual Studio 2015 и язык программирования C#. [5, 8, 12]

Для приложения Generator of tests была подключена библиотека Microsoft.Office.Interop.Excel [17], что позволило работать с excel файлами.

Для написания автоматических тестов был выбран инструмент для автоматизированного тестирования web-приложений Selenium WebDriver. Были подключены следующие библиотеки: OpenQA.Selenium, OpenQA.Selenium.Chrome, OpenQA.Selenium.Firefox, OpenQA.Selenium.IE, OpenQA.Selenium.Support.UI. С их помощью стало возможным управлять браузерами Google Chrome, Fierfox и Internet Explorer. А также, для распараллеливания заполнения значений на форме была подключена библиотека System.Threading.

В рамках автоматизации тестирования web-приложений были разработаны следующие программные продукты: приложение для генерации тестовых наборов и автоматические тесты с помощью Selenium WebDriver.

3.1. Приложение для генерации тестовых наборов

Приложение предназначено для генерации тестовых наборов на основе полученных условий (факторов) и значений для каждого из них. Количество факторов и значений для них может быть в следующих пределах: $[2; +\infty)$.

На выходе программа генерирует тестовые наборы. Каждый тестовый набор записывается в отдельный файл, который содержит в себе набор тестовых сценариев, созданный с использованием выбранного подхода.

3.1.1. Описание классов приложения для генерации тестовых наборов

Приложение состоит из семи классов и одного интерфейса. (см рис. 3.1)

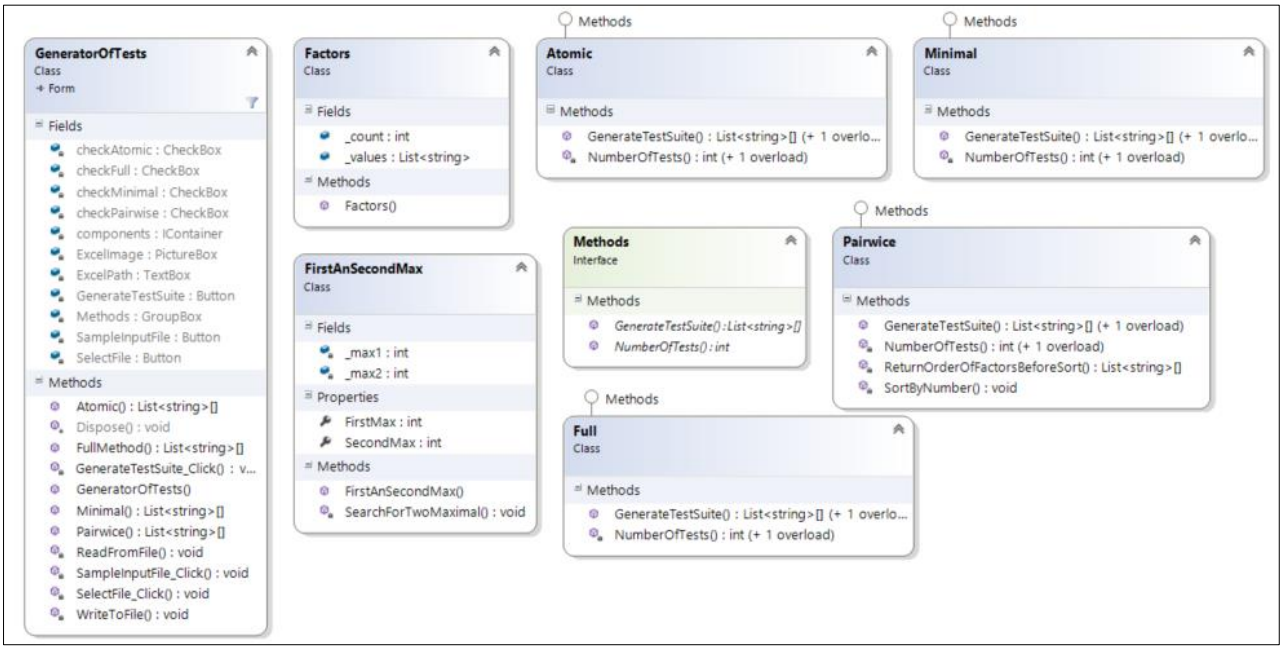


Рис. 3.1. Диаграмма классов приложения Generator of tests

Класс Factors имеет два поля: _count и _values. Первое содержит в себе количество переменных для фактора с тем же индексом, а второе все значения фактора того же индекса. Данный класс имеет всего лишь один метод, который заполняет переменные необходимыми значениями.

Класс FirstAndSecondMax имеет два поля: max1 и max2. Этот класс содержит в себе первое и второе максимальное число значений среди всех факторов. Он необходим, чтобы посчитать количество тестов для метода Pairwise. Класс FirstAndSecondMax содержит два параметра, которые возвращают первое и второе максимальное значение, но задать их нельзя. Класс имеет два метода: SearchForTwoMaximal и FirstAndSecondMax. Первый из них ищет первый и второй максимум и является приватным методом для данного класса, а второй присваивает значения переменным.

В интерфейсе Methods объявлены два метода, которые реализованы в следующих классах: Atomic, Minimal, Full и Pairwise.

Первый метод NumberOfTests предназначен для подсчета количества тестов. На вход принимает массив факторов со значениями и количество переменных в факторах (массив Factors) и возвращает число тестов, которое будет сгенерировано при применение выбранного подхода. Для каждого подхода используется свой алгоритм для подсчета количества тестов, поэтому данный метод реализован в каждом классе метода.

Второй метод GenerateTestSuite содержит в себе реализацию алгоритма для нахождения тестового набора. Метод на вход принимает массив факторов со значениями и количество переменных в факторах (массив Factors) и возвращает полученный тестовый набор. Для каждого из подходов были разработаны свои алгоритмы. (см. код нахождения тестовых наборов для подходов Full, Minimal и Atomic в приложениях 1, 2 и 3)

В классе Pairwise имеются дополнительные методы, которые не объявлены в интерфейсы, так как они применимы лишь для этого подхода: SortByNumber и ReturnOrderOfFactorsBeforeSort.

Метод SortByNumber принимает на вход массив факторов со значениями и количество переменных в факторах (массив Factors) и сортирует его по количеству переменных в факторе в порядке убывания.

Метод ReturnOrderOfFactorsBeforeSort принимает на вход полученный тестовый набор и массив факторов до сортировки и возвращает тестовый набор, в котором все факторы на тех местах, что и до сортировки SortByNumber. (см. код нахождения тестовых наборов для подхода Pairwise в приложении 4)

Класс GeneratorOfTests является центральным классом, в котором расположены методы, которые вызывают другие методы и классы для создания тестов, а также находятся компоненты, расположенные на форме для удобства использования программы.

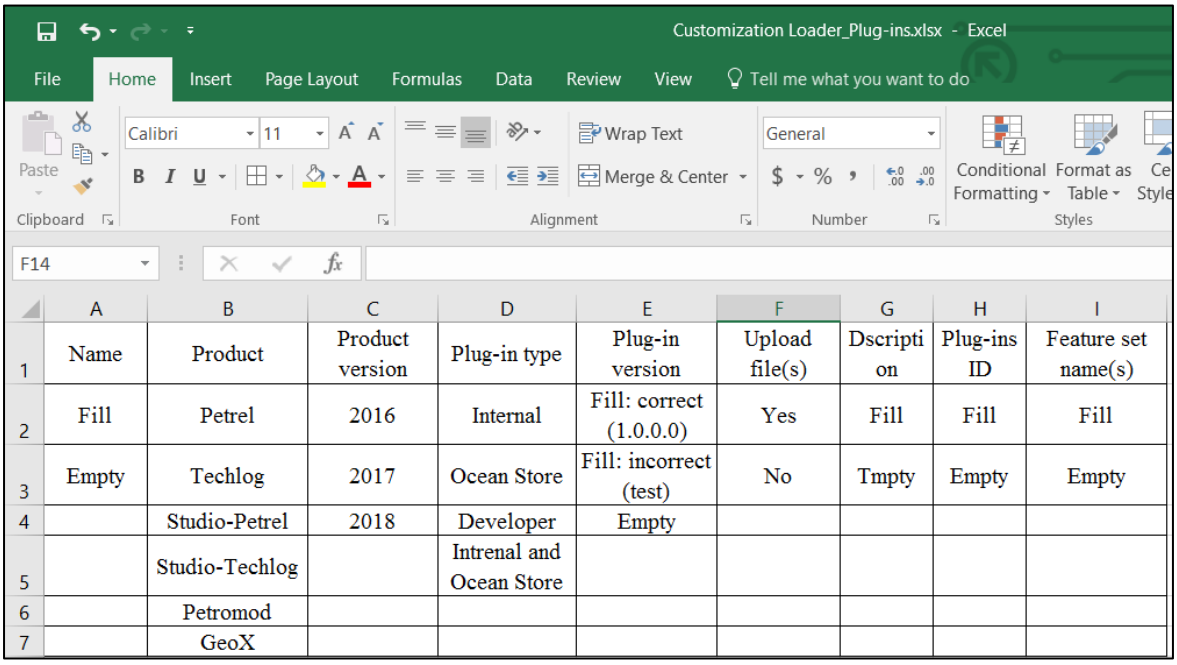
Метод SampleInputFile выгружает пример файла, который должен подаваться на вход программе.

Метод ReafFromFile считывает из входного excel файла факторы и их значения и записывает их в массив.

Метод WriteToFile генерирует excel файл с полученными тестовыми наборами для каждого из выбранных подходов.

3.1.2. Пример работы приложения

На вход приложению подается Excel документ, из которого считывается набор факторов и их переменных. (см. Рис. 3.2 и 3.3)



	A	B	C	D	E	F	G	H	I
1	Name	Product	Product version	Plug-in type	Plug-in version	Upload file(s)	Dscripti on	Plug-ins ID	Feature set name(s)
2	Fill	Petrel	2016	Internal	Fill: correct (1.0.0.0)	Yes	Fill	Fill	Fill
3	Empty	Techlog	2017	Ocean Store	Fill: incorrect (test)	No	Tmpty	Empty	Empty
4		Studio-Petrel	2018	Developer	Empty				
5		Studio-Techlog		Intrenal and Ocean Store					
6		Petromod							
7		GeoX							

Рис. 3.2. Пример входного файла

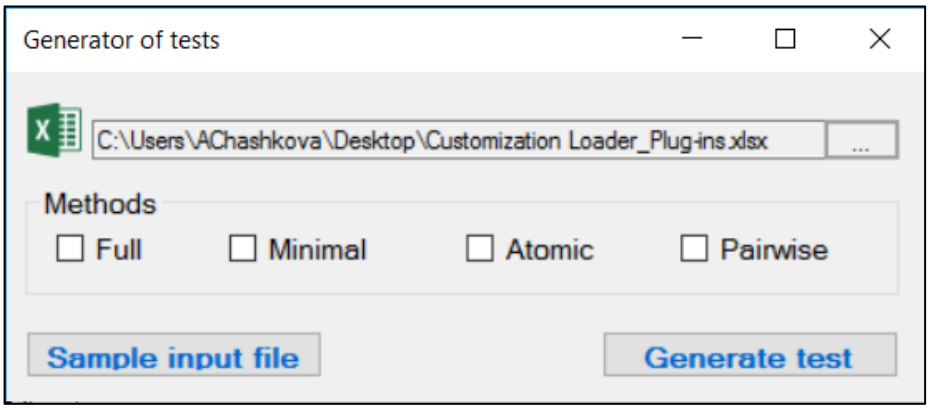


Рис. 3.3. Окно приложения для генерации тестового набора

При нажатии на кнопку Sample input file пользователь получает файл, формат которого соответствует тому, в каком формате должен быть загружен файл.

Пользователь имеет возможность выбрать с помощью какого подхода он хочет сгенерировать тестовый набор: это может быть, как один метод, так и сразу все четыре. (см. рис. 3.4.)

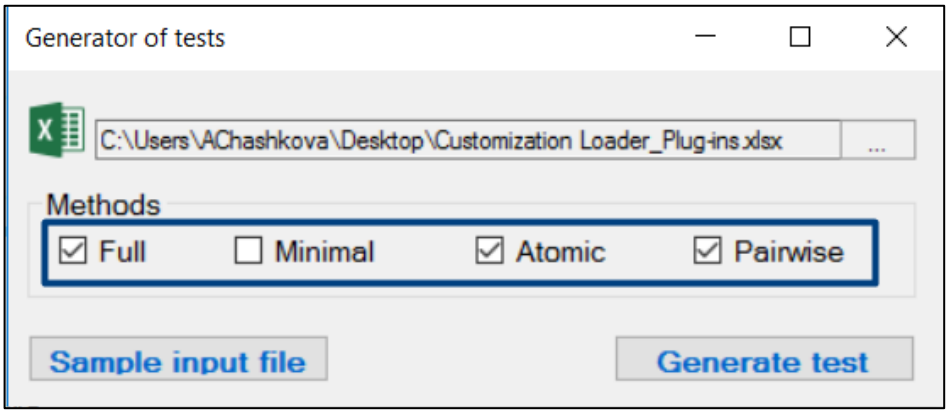


Рис. 3.4. Пример выбора подходов

После того, как методы выбраны, пользователь может сгенерировать тестовые наборы, нажав на кнопку Generated test. Указывает папку, в которую хочет сохранить тестовые наборы и на выходе имеет документ с тестовыми наборами. Под каждый метод генерируется отдельный файл. (см. рис. 3.5.) Каждый файл содержит набор тестовых сценариев в зависимости от выбранного подхода. (см. рис. 3.6)

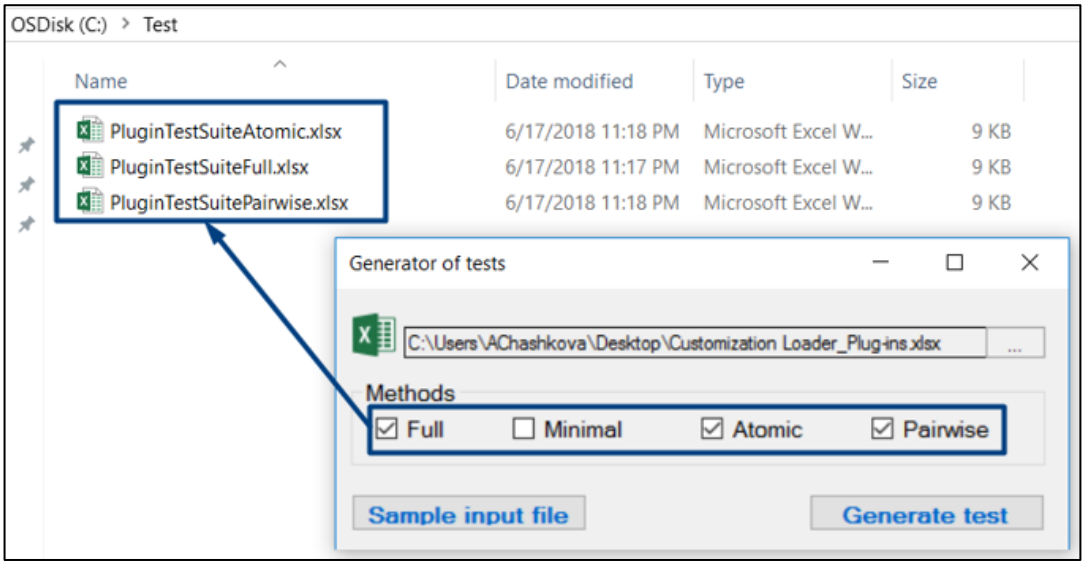
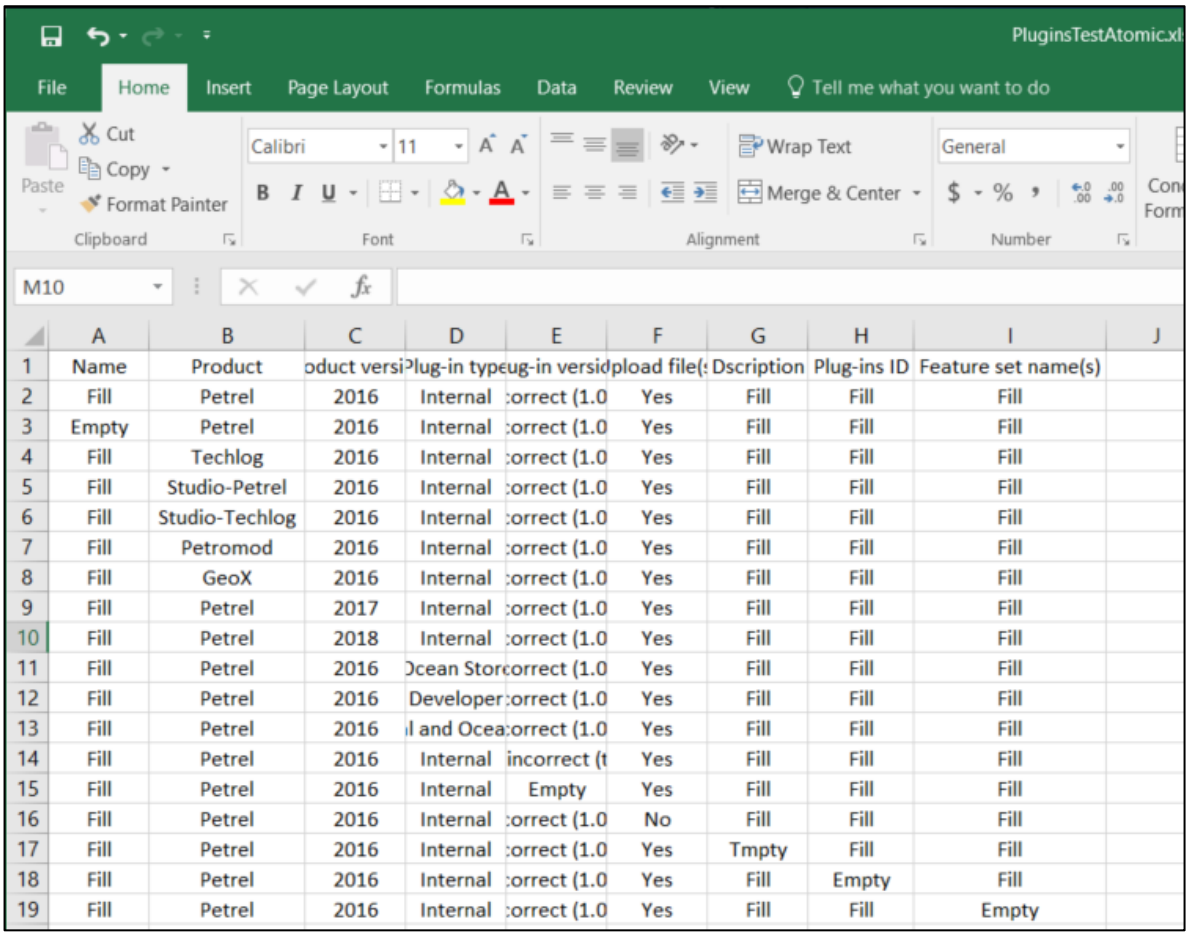


Рис. 3.5. Результат работы приложения



	A	B	C	D	E	F	G	H	I
1	Name	Product	oduct versi	Plug-in type	ug-in versi	pload file(s	Dscription	Plug-ins ID	Feature set name(s)
2	Fill	Petrel	2016	Internal	correct (1.0	Yes	Fill	Fill	Fill
3	Empty	Petrel	2016	Internal	correct (1.0	Yes	Fill	Fill	Fill
4	Fill	Techlog	2016	Internal	correct (1.0	Yes	Fill	Fill	Fill
5	Fill	Studio-Petrel	2016	Internal	correct (1.0	Yes	Fill	Fill	Fill
6	Fill	Studio-Techlog	2016	Internal	correct (1.0	Yes	Fill	Fill	Fill
7	Fill	Petromod	2016	Internal	correct (1.0	Yes	Fill	Fill	Fill
8	Fill	GeoX	2016	Internal	correct (1.0	Yes	Fill	Fill	Fill
9	Fill	Petrel	2017	Internal	correct (1.0	Yes	Fill	Fill	Fill
10	Fill	Petrel	2018	Internal	correct (1.0	Yes	Fill	Fill	Fill
11	Fill	Petrel	2016	Ocean Stor	correct (1.0	Yes	Fill	Fill	Fill
12	Fill	Petrel	2016	Developer	correct (1.0	Yes	Fill	Fill	Fill
13	Fill	Petrel	2016	l and Ocea	correct (1.0	Yes	Fill	Fill	Fill
14	Fill	Petrel	2016	Internal	incorrect (t	Yes	Fill	Fill	Fill
15	Fill	Petrel	2016	Internal	Empty	Yes	Fill	Fill	Fill
16	Fill	Petrel	2016	Internal	correct (1.0	No	Fill	Fill	Fill
17	Fill	Petrel	2016	Internal	correct (1.0	Yes	Tmpty	Fill	Fill
18	Fill	Petrel	2016	Internal	correct (1.0	Yes	Fill	Empty	Fill
19	Fill	Petrel	2016	Internal	correct (1.0	Yes	Fill	Fill	Empty

Рис. 3.6. Набор тестов для тестирования создания плагинов созданный с помощью метода атомарных проверок

3.2. Разработка автоматических тестов для тестирования сервисов Customization Loader и Image Creation

3.2.1. Customization Loader и Image Creation

Customization Loader – это сервис, предоставляющий UI для хранения расширений, которые в дальнейшим пользователь сможет использовать для создания образа виртуальной машины.

С помощью данного сервиса пользователь может легко добавить ресурсы, конфигурации, плагины и любые другие приложения, которые могут быть установлены на windows, но не разрабатываются компанией Schlumberger. При создании некоторые поля заполняются автоматически и некоторые вручную (см. Рис. 3.7). А также пользователь может просмотреть список всех созданных им продуктов, отредактировать или удалить их (см. Рис. 3.8).

Customizations Loader | Configurations | Resources | Plug-ins | Windows Apps | Anna Chashkova

Create new plug-in

ID* petrel-plugin-scal

Name* SCAL

Product* Petrel

Product version* 2016

Plug-in type* Ocean Store

Plug-in version* 2018.1

Plug-in ID(s) Add plug-in ID(s)

Feature set name(s) Add feature set name(s) for the plug-in.

Description Enter description of the plug-in

Media folder* gs://customizations-sis-lift-and-shift-stage/petrel-plugin-scal/2016+2018.1

Media Folder content

No files

Upload files Add files

Name	Size	Progress	Status	Actions
No files				

Рис. 3.7. Добавление плагина

Customizations Loader | Configurations | Resources | Plug-ins | Windows Apps | Anna Chashkova

Plug-ins

Filters

Product: Any | Product Version: Any | Plug-in type: Any | Upload status: Any | Apply

	ID	Product	Product version	Name	Version	Type	Upload status	Last updated	Actions
+	techlog-plugin-fullimage-computation	techlog	2017.1	FullImage Computation	3.0	Internal and Ocean Store	Completed	02/12/2018 12:33 GMT	
+	techlog-plugin-recall	techlog	2017.1	Recall	1.0	Internal and Ocean Store	Completed	04/16/2018 10:19 GMT	
+	petrel-plugin-hydraulic-fracture-modeler	petrel	2017.1	Hydraulic Fracture Modeler	2017.1.0	Internal and Ocean Store	Completed	02/12/2018 12:32 GMT	
+	techlog-plugin-cmrstandardstation	techlog	2017.1	CmrStandardStation	2.0.5	Internal and Ocean Store	Completed	02/12/2018 12:34 GMT	
+	techlog-plugin-cmrstacking	techlog	2017.1	CmrStacking	2.0.5	Internal and Ocean Store	Completed	02/12/2018 12:34 GMT	
+	techlog-plugin-petrography	techlog	2017.1	Petrography	1.0.3	Internal and Ocean Store	Completed	02/12/2018 12:33 GMT	

Рис. 3.8. Просмотр добавленных плагинов

Image Creation – это сервис, предоставляющий UI для создания образов виртуальных машин, которые в дальнейшим пользователь сможет использовать для создания виртуальной машины. При открытии данного сервиса пользователь попадает на стартовую страницу, где может создать образ (см. Рис. 3.9).

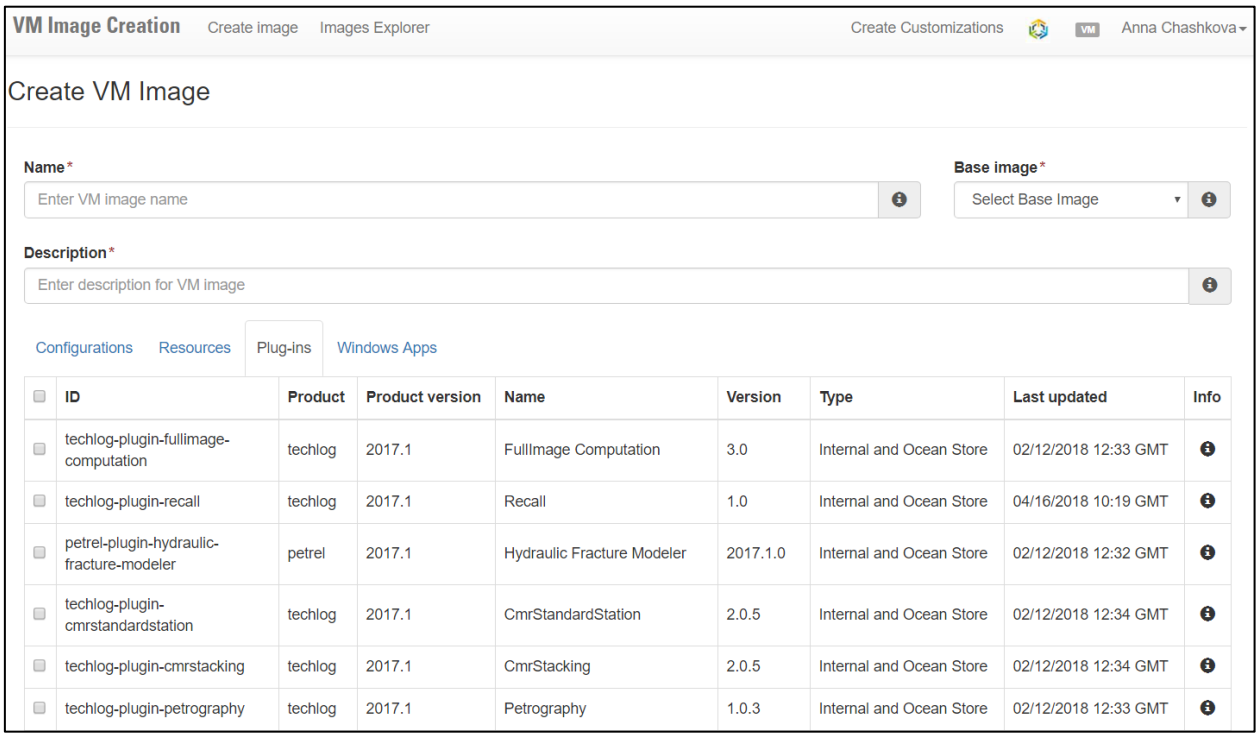


Рис. 3.19. Стартовая страница Image Creation

С помощью данного сервиса пользователь может создать свой собственный образ виртуальной машины, на котором будет все, что ему необходимо. После использования пользователь может удалить созданный образ. А также, может посмотреть какие образы уже существуют и что на них имеется (см. Рис. 3.10).

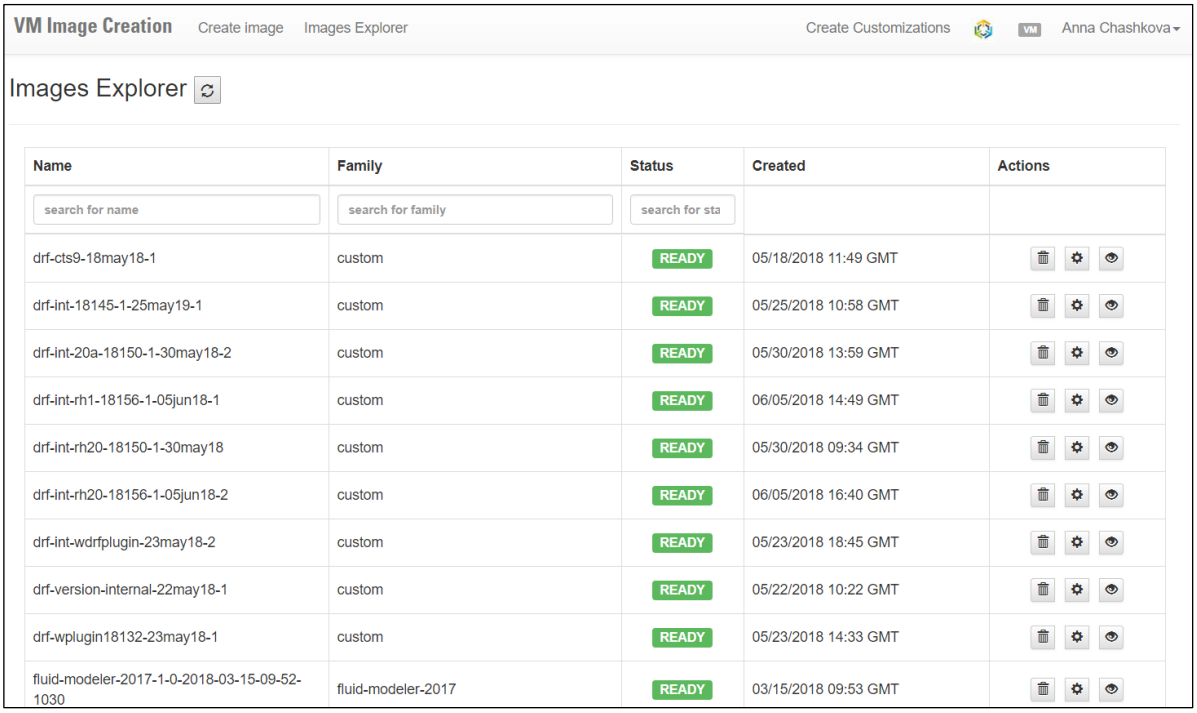


Рис. 3.10. Все образы, созданные пользователями

3.2.2. Создание тестовых наборов для тестирования сервисов Customization Loader и Image Creation

Всего было сгенерировано восемь тестовых наборов: Portals_Registration, Customization_Loader, CL_Create_Configurations, CL_Create_Resources, CL_Create_Plug-ins, CL_Create_WindowsApps, Image_Creation, IC_Create_Image. Для каждого из них был выбран тот или иной подход.

Для тестового набора Portals_Registration был выбран метод попарных проверок, т.к. нет смысла делать полный перебор, но сочетание пар параметров могут повлиять на результат работы. Всего факторов 3, а разброс значений для них от 2 до 3 (см. Таблица 5). С помощью метода был получен тестовый набор с 6 тестами.

Таблица 5.

Набор значений для тестирования регистрации на порталах

Portal	Browser	User
Customization Loader	Google Chrome	Schlumberger
Image Creation	Inetrnet Explorer	External
	Mozilla	

Для тестового набора Customization_Loader был выбран метод полного перебора т.к. все сочетания значений факторов необходимо проверить, чтобы удостоверится в работе портала. Всего факторов 3, а разброс значений для них от 3 до 4 (см. Таблица 6). С помощью метода был получен тестовый набор с 36 тестами.

Таблица 6.

Набор значений для тестирования портала Customization Loader

Browser	**Page**	**Actions**
Google Chrome	Configurations	Add
Mozilla	Resources	Edit
IE	Plug-ins	Delete
	Windows Apps	

Для тестовых наборов CL_Create_Configurations и CL_Create_Resources был выбран метод минимальных проверок т.к. значения факторов никак не повлияют друг на друга при работе, а значит их все можно проверять параллельно. Данные проверки имеют одинаковый набор факторов и значений для них. Всего факторов 5, а разброс значений для них от 2 до 6 (см. Таблица 7). С помощью метода был получен тестовый набор с 6 тестами для каждой из страниц.

Таблица 7.

Набор значений для тестирования создания Configurations и Resorces

Name	**Product**	**Product Version**	**Description**	**Upload files**
Fill	Petrel	2016	Fill	Yes
Empty	Techlog	2017	Empty	No
	Studio-Petrel	2018		
	Studio-Techlog			
	Petromod			
	GeoX			

Для тестового набора CL_Create_Plug-ins был выбран метод минимальных проверок т.к. значения факторов никак не повлияют друг на друга при работе, а значит их все можно проверять параллельно. Всего факторов 9, а разброс значений для них от 2 до 6 (см. Таблица 8). С помощью метода был получен тестовый набор с 6 тестами.

Таблица 8.

Набор значений для тестирования создания Plug-ins

Name	**Product**	**Product version**	**Plug-in type**	**Plug-in version**	**Upload file(s)**	**Dscription**	**Plug-ins ID**	**Feature set name(s)**
Fill	Petrel	2016	Internal	Fill: correct (1.0.0.0)	Yes	Fill	Fill	Fill
Empty	Techlog	2017	Ocean Store	Fill: incorrect (test)	No	Tmpty	Empty	Empty
	Studio-Petrel	2018	Developer	Empty				
	Studio-Techlog		Intrenal and Ocean Store					
	Petromod							
	GeoX							

Для тестового набора CL_Create_WindowsApps был выбран метод минимальных проверок т.к. значения факторов никак не повлияют друг на друга при работе, а значит их все можно проверять параллельно. Всего факторов 4, а разброс значений для них от 2 до 3 (см. Таблица 9). С помощью метода был получен тестовый набор с 3 тестами.

Таблица 9.

Набор значений для тестирования создания Windows Apps

Name	**Version**	**Description**
Fill	Fill: correct (1.0.0.0)	Fill
Empty	Fill: incorrect (test)	Empty
	Empty	

Для тестового набора Image_Creation был выбран метод полного перебора т.к. все сочетания значений факторов необходимо проверить, чтобы удостоверится в работе портала. Всего факторов 2, а разброс значений для них от 3 до 4 (см. Таблица 10). С помощью метода был получен тестовый набор с 12 тестами.

Таблица 10.

Набор значений для тестирования портала Image Creation

Browser	Actions
Google Chrome	Create VM
Inetrnet Explorer	Delete VM
Mozilla	Show details
	Base image assignments

Для тестового набора IC_Create_Image был выбран метод атомарных проверок т.к. с помощью него можно легко локализовать ошибки и провести необходимое количество тестов за оптимальное время. Всего факторов 4, а разброс значений для них от 2 до 6 (см. Таблица 11). С помощью метода был получен тестовый набор с 9 тестами.

Таблица 11.

Набор значений для тестирования создания Image

Name	**Base image**	**Description**	**Additional products**
Fill	Intrenal	Fill	All
Empty	Standart	Empty	Configuration
			Plug-ins
			Resources
			Windows App
			No

3.2.3. Описание классов приложения для проведения автоматического тестирование порталов Customization Loader и Image Creation

В разработке порталов Customization Loader и Image Creation принимают участие несколько команд из разных стран. API разрабатывается в команде, которая работает в Хьюстоне и наш сервис от него зависит, из-за чего любое изменения в API может повлиять на работу нашего сервиса. Поэтому, порой возникают ошибки из-за изменения одной из команд, а из-за невозможности постоянно быть на связи ошибки обнаруживаются поздно и на их исправление остается мало времени, а проверять весь функционал каждый день – не эффективно и затратно по времени. Поэтому было решено написать автоматические тесты, которые будут запускаться каждый день, а от человека необходимо будет лишь проверить результат их работы.

Было разработано приложение, которое содержит юнит-тесты для тестирования функционала портала по подобию работы с ним конечного пользователя. Приложение состоит из 17 классов (см. Рис. 3.11).

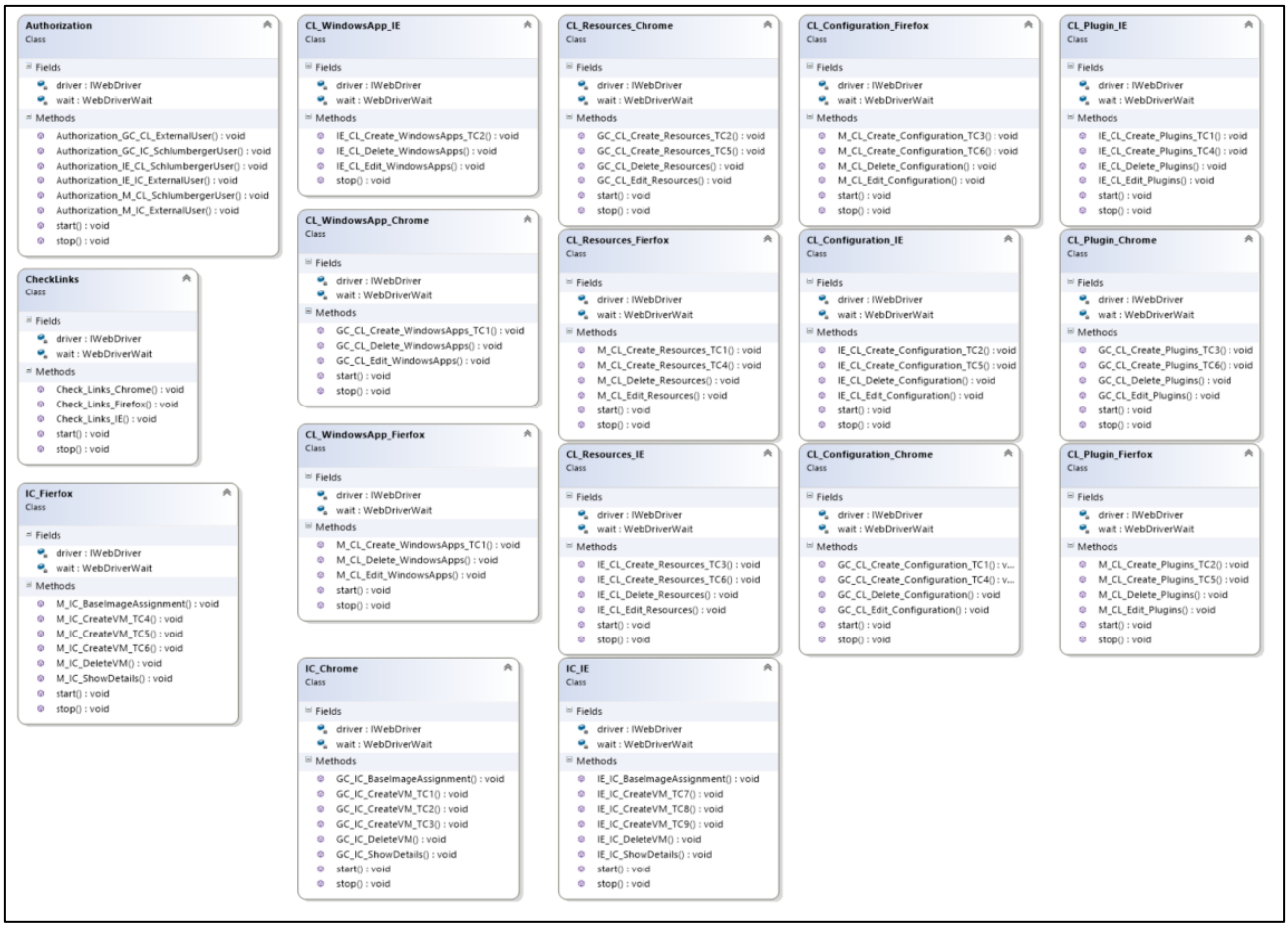


Рис. 3.11. Диаграмма классов приложения с набором автоматических тестов

Всего было написано 72 автоматических теста. Все они сгруппированы по классам, чтобы с легкостью можно было найти любой из них. Каждый класс содержит методы start и stop. Метод start запускает web driver для нужного браузера и инициализирует ожидание с заданным временем, что в тестах позволяет не устанавливать каждый раз ожидание, так как оно будет срабатывать автоматически: тест выжидает указанное время, либо продолжает работу, как только находит указанный элемент. Метод stop, который закрывает браузер по окончанию теста.

В классе Authorization находятся тесты, которые проверяют авторизацию пользователя на разных браузерах.

В классе CheckLinks содержатся проверки на корректность открытия ссылок и их работоспособность на различных браузерах.

Во всех остальных классах находятся тесты, которые проверяют те или иные возможности страницы (например: создание, удаление, редактирование и т.д.) на различных браузерах (пример автоматических тестов см. в Приложениях 5 и 6).

3.2.4. Автоматические тесты для тестирования сервисов Customization Loader и Image Creation

Тесты выглядят как обычные Unit tests и запускаются соответственно, но благодаря Selenium Web Driver стало возможным управлять браузерами и выполнять в них какие-либо действия.

Для более полного покрытия тестами были использованы ранее созданные тестовые наборы, которые обеспечивают максимальное покрытие за оптимальное время (см. Рис. 3.12).

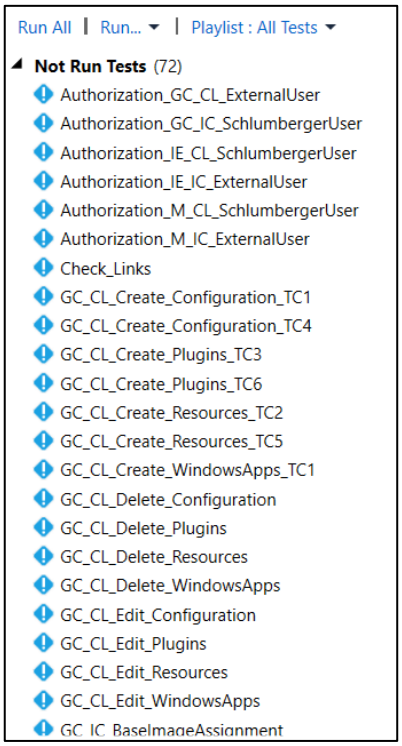


Рис. 3.12. Автоматические тесты для сервисов

Все тесты разработаны для тестирования в следующих браузерах: Google Chrome, Mozilla и Internet Explorer.

При запуске тестов запускается окно браузера и все выполняемые действия можно увидеть. На вход тесту подается ссылка с сайтом, на котором будут выполнятся действия, после чего начинается поиск элементов и выполнение действий с ними.

Автоматические тесты выполняются быстрее, чем если бы, то же самое делал человек. Выполнение данных тестов занимает всего около 4 – 4.5 часов, так как создание одного образа выполняется от 20 до 30 минут, поэтому эти тесты занимают много времени, в то время как все остальные проходят за час в общей сложности. Несмотря на то, что даже автоматические тесты занимают несколько часов, ручное тестирование занимает еще больше времени, а в случае с автоматическими тестами, пока они выполняются, инженер по качеству может выполнять другие задачи.

Ниже показан один из результатов разработанных автоматических тестов. (см. Рис. 3.13) Почти все тесты имеют статус Passed и лишь один Failed. В данном случае он не был выполнен успешно потому, что по времени не успел загрузить страницу.

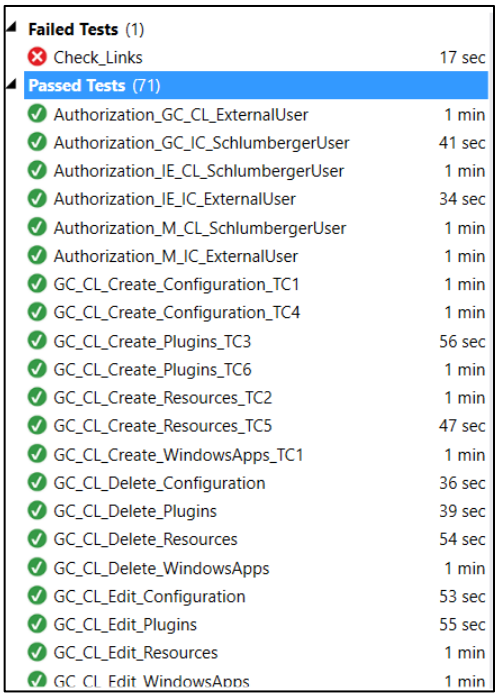


Рис. 3.13. Результат автоматических тестов

Заключение

В ходе данной работы были изучены известные методы (черного, белого и серого ящиков), виды (функциональное, стрессовое, нагрузочное, тестирование производительности, стабильности, удобства использования, безопасности, совместимости) и формы (модульное, системное, интеграционное и приемочное) тестирования программного обеспечения.

Подробно было изучено тестирование web-приложений. Для написания автоматических тестов компанией Schlumberger был выбран и изучен инструмент для автоматизированного тестирования Selenium.

В процессе работы были рассмотрены следующе 4 подхода: полный перебор, метод минимальных проверок, метод атомарных проверок и метод попарных проверок. Данные подходы были описаны, а также, на основании полученных знаний о них, были выведены алгоритмы для создания тестовых наборов и оценена сложность каждого из них.

На основании созданных алгоритмов была написана программа для генерации тестовых наборов, которая успешно внедрена в компанию Schlumberger и используется для создания тестовых наборов, как для web-приложений, так и для других программ.

С помощью созданного приложения были сгенерированы тестовые наборы для тестирования сервисов Customization Loader и Image Creation, а на основании полученных наборов были разработаны автоматические тесты с использованием инструмента для автоматизированного тестирования Selenium WebDriver.

Таким образом, все поставленные задачи выполнены.

Разработанное программное обеспечение позволило проверять основные функции порталов без больших временных затрат инженера по качеству. Таким образом, процесс проверки web-приложений автоматизирован, а значит цель достигнута.

Список используемой литературы

1. Aplana Quality Services [электронный ресурс]: Приемочное тестирвоание, режим доступа: свободный, последняя дата обращения: 02.05.18. http://aplana.ru/services/testing/priemochnoe-testirovanie;
2. Bach J. and Shroeder. P. Pairwise testing - a best practice that isn't. In Proceedings of the 22nd Pacific Northwest Software Quality Conference, pages 180–196, 2004.;
3. Burr K. and Young W. Combinatorial test techniques: Table-based automation, test generation, and test coverage. In Proceedings of the International Conference on Software Testing, Analysis, and Review (STAR), San Diego CA, 1998.;
4. Guide to the Software Engineering Body of Knowledge, 2004 version - Software testing, 3 выпуск, 2004 – 204 с.;
5. Dan Clark, Beginning C# Object-Oriented Programming, Издательство: Apress, 2013 – 384с.;
6. Unmesh Gundecha, Selenium Testing Tools Cookbook/ U. Gundecha – 1 выпуск, 2012г, 326 с.;
7. Wikispaces [электронный ресурс]: Ручное тестирование, режим доступа: свободный, последняя дата обращения: 25.05.16. http://tpl-it.wikispaces.com/;
8. Введение в модульное тестирование для c# проектов в среде MonoDevelop [электронный ресурс]: 2012 – Режим доступа: свободный, последняя дата обращения: 28.04.2018: https://habrahabr.ru/post/181255/;
9. Дж. Макконелл Анализ алгоритмов. Активный обучающий подход. — 3-е дополненное издание. М: Техносфера, 2009. – 416с.;
10. Интеграционное тестирование [электронный ресурс] – 2008 – 2018 – Режим доступа: http://www.protesting.ru/testing/levels/integration.html, свободный – Загл. с экрана. (дата обращения: 03.06.2016);

11. Куликов С., Тестирование программного обеспечения. Базовый курс. – 2 издание, 2018 г., 296 с.;

12. Культин Н. Microsoft Visual C# в задачах и примерах (+ CD-ROM). БХВ-Петербург – 2012 г., 314 с.;

13. Лекция 10. Документирование и оценка индустриального тестирования – Интуит: национально открытый университет –, 2016 - 3с.;

14. Миллер, Р. Последовательные и параллельные алгоритмы: Общий подход / Р. Миллер, Л. Боксер ; пер. с англ. — М. : БИНОМ. Лаборатория знаний, 2006. — 406 с.;

15. Савин Р. Тестирование Дот Ком. - М.: Дело, 2007. – 312 с.;

16. Системное тестирование [электронный ресурс] – 2008-2016 – Режим доступа: http://www.protesting.ru/testing/levels/system.html, свободный – Загл. с экрана. (дата обращения: 04.06.2016);

17. Сравнение библиотек для работы с файлами Misrosoft Excel [электронный ресурс]: Сосонных О. - Сравнение библиотек для работы с файлами Misrosoft, 2018 г. – Режим доступа: свободный, дата обращения: 20.05.2018: http://www.compdoc.ru/prog/pascal/compar_libraries_for_work_excel/;

18. Степанченко И. В. Метод тестирования программного обеспечения – Волгоград, Учеб. пособие / ВолгГТУ, 2006. – 74 с.;

19. Стотлемайер Д. Тестирование Web-приложений. Средства и методы для автоматизированного и ручного тестирования программного обеспечения Web-сайтов : пер. с англ / Д. Стотлемайер. - Москва: КУДИЦ-ОБРАЗ, 2003., 237 с.;

20. Тестирование. Фундаментальная теория [электронный ресурс]: Мищевский Г. - Тестирование. Фундаментальная теория, 2016, - Режим доступа: свободный, дата обращения: 20.05.2016: https://habrahabr.ru/post/279535/;

21. Томас Х. Кормен Алгоритмы: вводный курс Томаса Х. Кормена = Algorithms Unlocked. — М.: «Вильямс», 2014. — 208 с.;

22. Учебно-метадические материалы для студентов кафедры АСОИУ [электронный ресурс]: Цыганенко В.Н. – Тема 6. Определение тестирования программного обеспечения, 2012 – Режим доступа: свободный, дата обращения: 20.05.2016: http://www.4stud.info/software-construction-and-testing/lecture7.html.

Приложение 1. Нахождение тестового набора с использованием метода полного перебора

```
class Full: Methods
    {
        int NumberOfTests(Factors[] F)
        {
            int count = 1;
            for (int i = 0; i < F.Count(); i++)
                count = count * F[i]._count;
            return count;
        }

        public List<string>[] GenerateTestSuite(Factors[] F)
        {
            int _NumberOfTests = NumberOfTests(F);

            List<string>[] T = new List<string>[_NumberOfTests];
            for (int i = 0; i < _NumberOfTests; i++)
                T[i] = new List<string>();

            int step = 0;
            for (int j = 0; j < F.Count(); j++)
            {
                step = 1;
                for (int m = 0; m <= j; m++)
                    step = step * F[m]._count;
                step = _NumberOfTests / step;

                for (int k = 1; k <= F[j]._count; k++)
                    for (int i = k * step - step; i < _NumberOfTests; i++)
                    {
                        for (int l = 0; l < step; l++)
                        {
                            T[i].Add(F[j]._values[k - 1]);
                            i++;
                        }
                        i = i + step * (F[j]._count - 1) - 1;
                    }
            }
            return T;
        }
    }
```

Приложение 2. Нахождение тестового набора с использованием метода минимальных проверок

```
class Minimal: Methods
    {
        int NumberOfTests(Factors[] F)
        {
            int count = 0;
            for (int i = 0; i < F.Count(); i++)
                if (count < F[i]._count)
                    count = F[i]._count;
            return count;
        }

        public List<string>[] GenerateTestSuite(Factors[] F)
        {
            int _NumberOfTests = NumberOfTests(F);

            List<string>[] T = new List<string>[_NumberOfTests];
            for (int i = 0; i < _NumberOfTests; i++)
                T[i] = new List<string>();

            for (int j = 0; j < F.Count(); j++)
                for (int k = 0; k < F[j]._count; k++)
                    for (int i = k; i < _NumberOfTests; i++)
                    {
                        T[i].Add(F[j]._values[k]);
                        if (i < F[j]._count)
                            i = i + F[j]._count - 1;
                        if (k != 0) break;
                    }
            return T;
         }
     }
```

Приложение 3. Нахождение тестового набора с использованием метода атомарных проверок

```
class Atomic: Methods
    {
        int NumberOfTests(Factors[] F)
        {
            int count = 0;
            for (int i = 0; i < F.Count(); i++)
                count = count + F[i]._count;
            count = count - F.Count() + 1;
            return count;
        }

        public List<string>[] GenerateTestSuite(Factors[] F)
        {
            int _NumberOfTests = NumberOfTests(F);

            List<string>[] T = new List<string>[_NumberOfTests];
            for (int i = 0; i < _NumberOfTests; i++)
                T[i] = new List<string>();

            for (int j = 0; j < F.Count(); j++)
                for (int i = 0; i < _NumberOfTests; i++)
                {
                    int numberOfUsedValues = 0;
                    for (int k = 0; k < j; k++)
                        numberOfUsedValues = numberOfUsedValues + F[k]._count;

                    if (i > numberOfUsedValues - j && i < numberOfUsedValues +
F[j]._count - j)
                        T[i].Add(F[j]._values[i + j - numberOfUsedValues]);
                    else
                        T[i].Add(F[j]._values[0]);
                }

            return T;
        }
    }
```

Приложение 4. Нахождение тестового набора с использованием метода попранных проверок

```
class Pairwice: Methods
    {
        int NumberOfTests(Factors[] F)
        {
            FirstAnSecondMax m = new FirstAnSecondMax(F);
            return m.FirstMax * m.SecondMax;
        }

        void SortByNumber(ref Factors[] F)
        {
            Array.Sort(F, (x, y) => y._count.CompareTo(x._count));
        }

        List<string>[] ReturnOrderOfFactorsBeforeSort(Factors[] F, List<string>[] T,
List<string>[] L, int count)
        {
            List<int> order = new List<int>();

            for (int i = 0; i < F.Count(); i++)
                for (int j = 0; j < F.Count(); j++)
                    if (L[i].Contains(F[j]._values.First()))
                        order.Add(j);

            List<string>[] Temp = new List<string>[count];
            for (int i = 0; i < count; i++)
                Temp[i] = new List<string>();

            for (int i = 0; i < F.Count(); i++)
                for (int j = 0; j < count; j++)
                    Temp[j].Add(T[j][order[i]]);

            return Temp;
        }

        public List<string>[] GenerateTestSuite(Factors[] F, List<string>[] L)
        {
            int _NumberOfTests = NumberOfTests(F);

            SortByNumber(ref F);

            List<string>[] T = new List<string>[_NumberOfTests];
            for (int i = 0; i < _NumberOfTests; i++)
                T[i] = new List<string>();

            for (int k = 1; k <= F[0]._values.Count(); k++)
                for (int i = k * F[1]._values.Count() - F[1]._values.Count(); i <
_NumberOfTests; i++)
                {
                    for (int l = 0; l < F[1]._values.Count(); l++)
                    {
                        T[i].Add(F[0]._values[k - 1]);
                        i++;
                    }
                    i = i + F[1]._values.Count() * (F[0]._values.Count - 1) - 1;
                }

            int countWithTheSameValues = 0;

            for (int j = 1; j < F.Count(); j++)
            {
```

```
                if (F[j - 1]._count == F[j]._count && j != 1)
                    countWithTheSameValues++;
                else countWithTheSameValues = 0;

                for (int i = 0; i < _NumberOfTests; i++)
                {
                    for (int k = 0; k < F[j]._count && i < _NumberOfTests; k++)
                    {
                        if (i >= (countWithTheSameValues - 1) * F[j]._count && i <
countWithTheSameValues * F[j]._count && countWithTheSameValues != 0)
                            T[i].Add(F[j]._values[(k + 1) % F[j]._count]);
                        else T[i].Add(F[j]._values[(k) % F[j]._count]);
                        i++;
                    }
                    i--;
                }
            }

            return ReturnOrderOfFactorsBeforeSort(F, T, L, _NumberOfTests);
        }
    }
```

Приложение 5. Проверка корректности всех ссылок на порталах Customization Loader и Image Creation

```
public void Check_Links()
    {
        start();
        driver.Url = "https://customizations-dot-pitc-shared-dev.appspot.com/#/";
        driver.Manage().Window.Maximize();
        driver.FindElement(By.ClassName("idp")).Click();
        var links = driver.FindElements(By.TagName("a"));
        for (int i = 0; i < links.Count; i++)
        {
            string title = links[i].Text;
            if (title.Length != 0)
                title = title.Remove(links[i].Text.Length - 1, 1);

            if ("Customizations Loader".Contains(title))
            {
                links[i].Click();
                Thread.Sleep(1000);
                if (!driver.FindElement(By.TagName("h1")).Text.ToLower().Contains("welcome!"))
                    NUnit.Framework.Assert.Fail(title + " doesn't contain" + driver.FindElement(By.TagName("h1")).Text);
                driver.SwitchTo().Window(driver.WindowHandles[0]);
                driver.FindElements(By.TagName("a"))[0].Click();
                Thread.Sleep(1000);
                links = driver.FindElements(By.TagName("a"));
            }
            else
            {
                for (int j = 0; j < 2 && i < links.Count - 1; j++, i++)
                {
                    links[i - j].Click();
                    Thread.Sleep(1000);
                    if (driver.FindElements(By.LinkText("Create Customizations")).Count == 0)
                    {
                        links[i + 1].Click();
                        Thread.Sleep(1000);
                        var t = driver.FindElements(By.TagName("h3"))[1].Text;
                        if (!t.ToLower().Contains(title.ToLower()))
                            NUnit.Framework.Assert.Fail(title + " doesn't contain " +
driver.FindElement(By.TagName("h3")).Text);
                    }
                    else
                    {
                        driver.FindElement(By.LinkText("Create Customizations")).Click();
                        Thread.Sleep(1000);
                        if (!driver.FindElement(By.TagName("h1")).Text.ToLower().Contains("welcome!"))
                            NUnit.Framework.Assert.Fail(title + " doesn't contain" +
driver.FindElement(By.TagName("h1")).Text);
                        j++; i--;
                    }
                    driver.FindElements(By.TagName("a"))[0].Click();
                    Thread.Sleep(1000);
                    links = driver.FindElements(By.TagName("a"));
                }
            }
        }
        System.Threading.Thread.Sleep(2000);
        stop();
    }
```

Приложение 6. Создание записи о плагине на портале Customization Loader

```
public void Pugin_Create_Chrome()
    {
        start();
        driver.Manage().Window.Maximize();
        driver.Url = "https://customizations-dot-pitc-shared-dev.appspot.com/#/";
        driver.FindElement(By.ClassName("idp")).Click();
        driver.FindElement(By.LinkText("Plug-ins")).Click();
        driver.FindElement(By.LinkText("Add new plug-in")).Click();
        System.Threading.Thread.Sleep(1000);
        driver.FindElement(By.CssSelector("span[uib-popover='Selected product is absolete, before update please
change product']")).Click();
        System.Threading.Thread.Sleep(2000);
        if (!driver.FindElement(By.TagName("small")).Displayed)
            NUnit.Framework.Assert.Fail("Error message is missing");
        NUnit.Framework.Assert.That(driver.FindElement(By.TagName("small")).Text == "This field is required");
        driver.FindElement(By.Name("name")).SendKeys("SCAL");
        driver.FindElement(By.Id("product")).Click();
        driver.FindElement(By.Id("product")).FindElements(By.TagName("option"))[1].Click();
        driver.FindElement(By.Id("productVersion")).Click();
        driver.FindElement(By.Id("productVersion")).FindElements(By.TagName("option"))[10].Click();
        driver.FindElement(By.Id("pluginType")).Click();
        driver.FindElement(By.Id("pluginType")).FindElements(By.TagName("option"))[1].Click();
        driver.FindElement(By.CssSelector("span[uib-popover='Selected product is absolete, before update please
change product']")).Click();
        System.Threading.Thread.Sleep(5000);
        stop();

    }
```