|  |
| --- |
| УТВЕРЖДАЮ  Профессор кафедры ЦТУТП, д.т.н.  \_\_\_\_ Э. К. Лецкий  « 24 » 03 2022 год |

**ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ**

**по дисциплине**

**«Проектирование информационных систем»**

студента группы УВА-411 Романова Даниила Николаевича

**Наименование темы:**

Разработка структуры и алгоритмов реализации гибридного фреймворка для автоматизации тестирования приложений.

**Необходимо изучить:**

1. Актуальные средства и инструменты для автоматизации тестирования.
2. Архитектуру проектов по автоматизации тестирования.
3. Паттерны проектирования информационных систем по автоматизации.

**Необходимо самостоятельно выполнить:**

1. Разработку архитектуры модулей фреймворка.
2. Реализовать алгоритмы для поиска элементов интерфейса.
3. Классификацию и структуру тестовых сценариев.
4. Разработку типовой модели автоматизированных тестов.

**Основные требования к проделываемой работе:**

1. Представить структуру программного обеспечения.
2. Использовать ГОСТ Р 7.32-2001 – для оформления пояснительной записки

**Источники информации:**

1. Джерард Месарош «Шаблоны тестирования xUnit. Рефакторинг кода тестов». – Москва: «Вильямс», 2009. – 832 с.
2. Эрик Фримен, Элизабет Фримен, Кэтти Сьерра, Берт Бейтс «Паттерны проектирования» – СПб.: «Питер», 2013 – 832 с.
3. <https://ru.selenide.org/> – фреймворк для автоматизированного тестирования веб-приложений на основе Selenium WebDriver.

**Материалы к защите:**

1. Заполненный, подписанный и утверждённый лист индивидуального задания на курсовой проект.
2. Пояснительная записка (отчёт).
3. Презентация о проделанной работе (только для случай комиссионной защиты)

|  |  |
| --- | --- |
| **Студент группы** **УВА-411** C:\Users\VTB\Pictures\Подпись.png | Романов Д.Н |
| **Руководитель курсового проекта** | Соймина Е.Я. |

* 1. ВЕДЕНИЕ В ПРЕДМЕТНУЮ ОБЛАСТЬ

В современном мире внедрение информационных технологий становится неотъемлемым требованием в различных областях человеческой деятельности.

Крупным IT компаниям приходится постоянно решать новые задачи, гибко подстраиваясь под требования рынка. Постоянно растёт потребность в разработке нового программного обеспечения. Решения находят быстро, но зачастую они не всегда оказываются наиболее эффективными.

Каждая новая программа (система), проходит несколько жизненно важных этапов разработки. Одним из этапов является тестирование программного обеспечения. Задачи организации тестирования программного обеспечения каждая компания решает по-своему, но проблема остаётся на сегодняшний день открытой.

Для проведения тестирования ПО требуется постоянно растущий штат специалистов и серьезные временные затраты, что оборачивается для компании большими убытками.

Проблема является наиболее актуальной для крупных компаний, в которых тестированию подлежит большой объем создаваемого программного обеспечения.

Цель исследования – сделать процесс тестирования программного обеспечения гибким, не ресурсоемким и удобным.

Задачи курсового:

- проанализировать существующие подходы к тестированию программного обеспечения выявить достоинства и недостатки;

- проанализировать пути автоматизации процесса тестирования

- предложить подход к решению проблемы с использованием средств автоматизации

1. Разработка архитектуры модулей фреймворка

Фреймворк состоит из трёх модулей.

* Функциональный модуль
* Связующий модуль
* Модуль тестовых сценариев

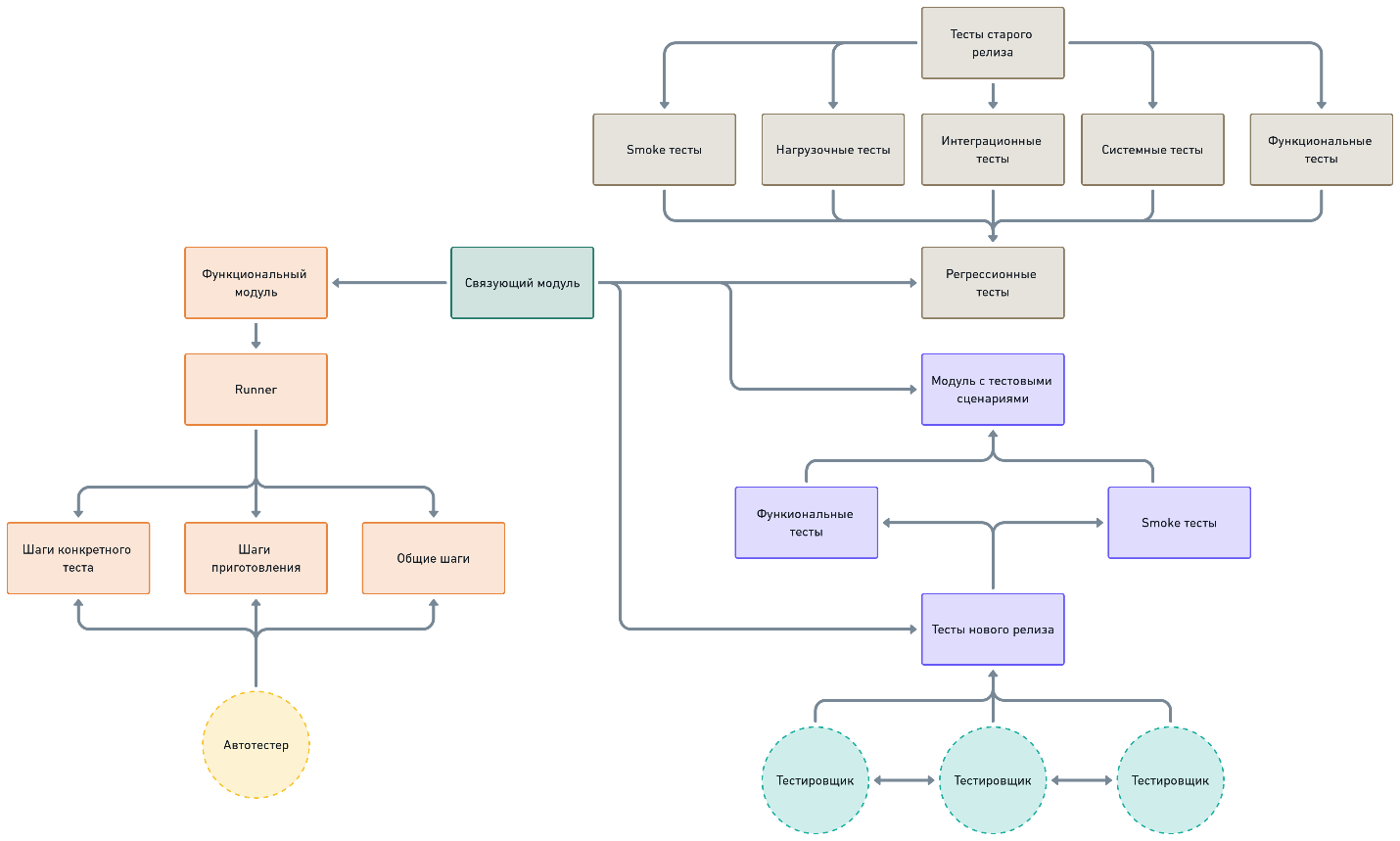


Рис. 1 – Структурная схема модулей

Модуль тестовых сценариев содержит файлы с описанием тестовых сценариев. В файлах описываются шаги и действия, выполняемые в тестах, а также наборы тест-кейсов.

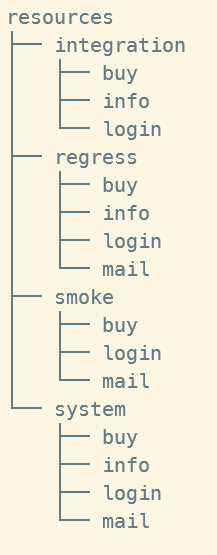


Рис. 2 – Структура директорий

Верхний уровень директорий — это тип тестовых сценариев. Они подразделяются на множество типов, например – system (системное тестирование) или regress (регрессионное тестирование).

Нижний уровень директорий – это функциональный модуль самого приложения. Например, buy (покупки в приложении) или login (авторизация в приложении).

Разделение тестов на папки является хорошей практикой в проектировании архитектуры тестовых сценариев.

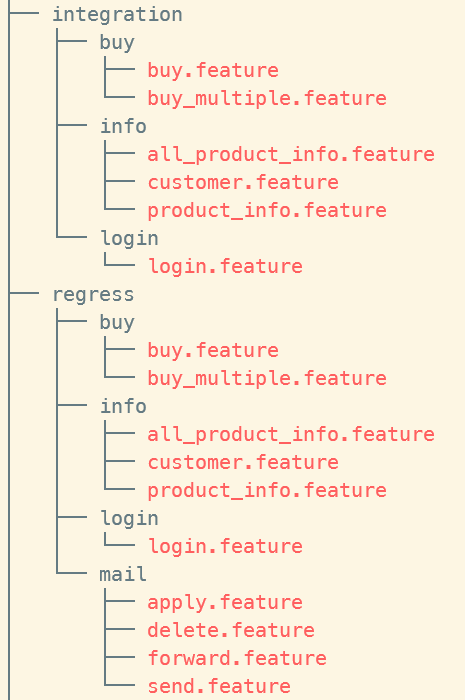


Рис. 3 – Структура файлов тестовых сценариев

Файлы в формате «.feature» содержат тест-кейсы и сценарии.



Рис. 4 – Файл с примером тестового сценария

Специальное слово «Feature» служит для объявления набора тестов, у него есть описание и название.

Слово «Scenario» означает объявление самого теста, имеет имя.

Слово «Then» означает шаг внутри теста, например, «Нажать на кнопку подтверждения» значит найти на странице кнопку типа «submit» и кликнуть на неё.

**Связующий модуль**

Связующий модуль реализован при помощи библиотек компании «Cucumber», в нём задействован функционал связи ключевых слов с кодом проекта. Реализация основана на имплементации ключевых слов в файлах .feature и их связи с реальными тестовыми методами через аннотации @Then.

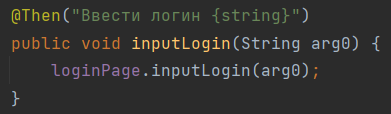


Рис. 5 – Реализация метода ввода логина

Cucumber ищет среди классов проекта методы помеченные аннотацией @Then и исполняет их, передавая необходимые аргументы или без таковых.

Таким образом, связующий модуль используется исключительно как сторонняя библиотека для реализации связи между описанием тестового сценария и исполнением реального кода проекта.

**Функциональный модуль**

Реализуется специалистом по автоматизации тестирования. Имеет следующую структуру папок.

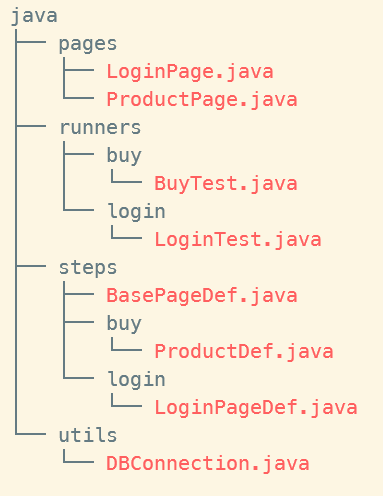


Рис. 6 – Структура классов

Директория pages содержит в себе классы, описывающие элементы web страниц и действия, которые можно производить над ними.

Директория runners содержит классы, отвечающие за запуск тестов выбранного файла .feature.

Директория steps содержит классы реализующие тестовые шаги

Директория utils содержит вспомогательные утилиты, которые упрощают работу со специфичными инструментами.

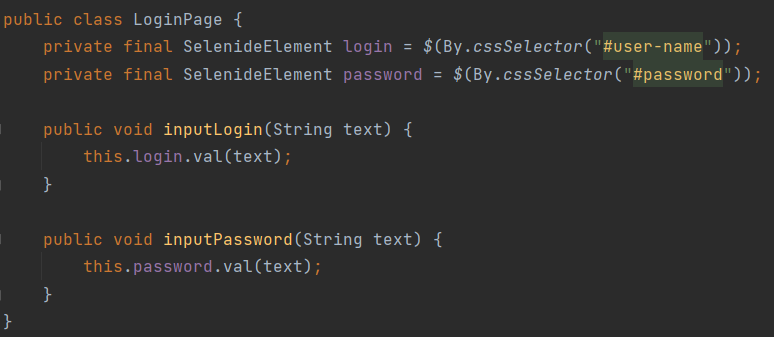


Рис. 7 – Структура Page Object

Класс LoginPage описывает страницу логина, где реализованы функции ввода логина и пароля пользователя.

Элементы были получены через свойства HTML страницы по уникальному идентификатору.

В классах паттерна Page Object реализуется интерфейс для работы с конкретной станицей по средствам кода, в данном примере используется страница логина.

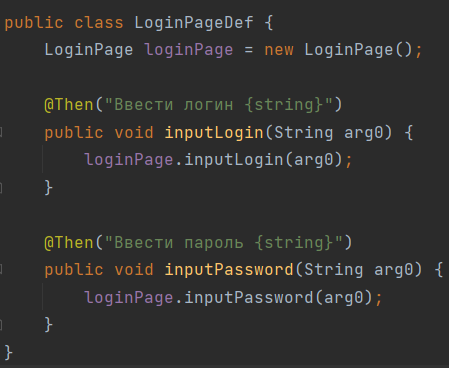


Рис. 8 – Структура тестовых шагов

Для реализации тестовых шагов используются объекты всех страниц, задействованных в тесте. В данном случае страница одна, поэтому используется один класс LoginPage.

В классах с тестовыми шагами реализуются только уникальные для каждого теста действия. Это необходимо для переиспользования уже написанного кода.

Все общие шаги выносятся в отдельный класс для реализации.



Рис. 9 – Структура тестовых шагов

Класс с общими шагами содержит самые популярные методы, например, openUrl, который запустит браузер со страницей тестируемого приложения.

1. Реализация алгоритмов для поиска элементов интерфейса.

Страница HTML представляет собой дерево объектов, у каждого из которых могут быть свои дочерние объекты. Иначе это называется «Document Object Model» (DOM).

XPath селекторы используются для быстрой навигации по элементам страницы, они представляют из себя полный путь или относительный путь до необходимого объекта, также у них есть множество специальных функций, например, функция «text» возвращает текст элемента.

Квадратные скобки означают порядковый номер вложенного элемента.

По мимо XPath существуют CSS селекторы, они считаются более надёжными из-за редко меняющихся стилей на страницах приложений.

Оба алгоритма поиска реализованы через обход дерева и считывания каждого элемента на пути. Алгоритм реализован следующим образом (Рис. 10).



Рис. 10 – Алгоритм поиска элементов DOM

Исходя из блок-схемы можно сделать вывод, что XPath/CSS селекторы работают по одному и тому же сценарию. Главное и весомое отличие CSS от XPath в том, что CSS селекторы не работают с текстом элементов, что делает их менее универсальными.

Все алгоритмы поиска элементов базируются на обходе дерева DOM, однако существуют некоторые функции селекторов, которые позволяют существенно сократить время поиска. Например, мы можем указать конкретный атрибут элемента, чтобы механизмы языка загружали только ту часть страницы, в которой они существуют.

Этот механизм сократит время на ожидании загрузки страницы в буфер.

1. Классификацию и структуру тестовых сценариев.

Тестовые сценарии необходимо разделять на несколько классов.

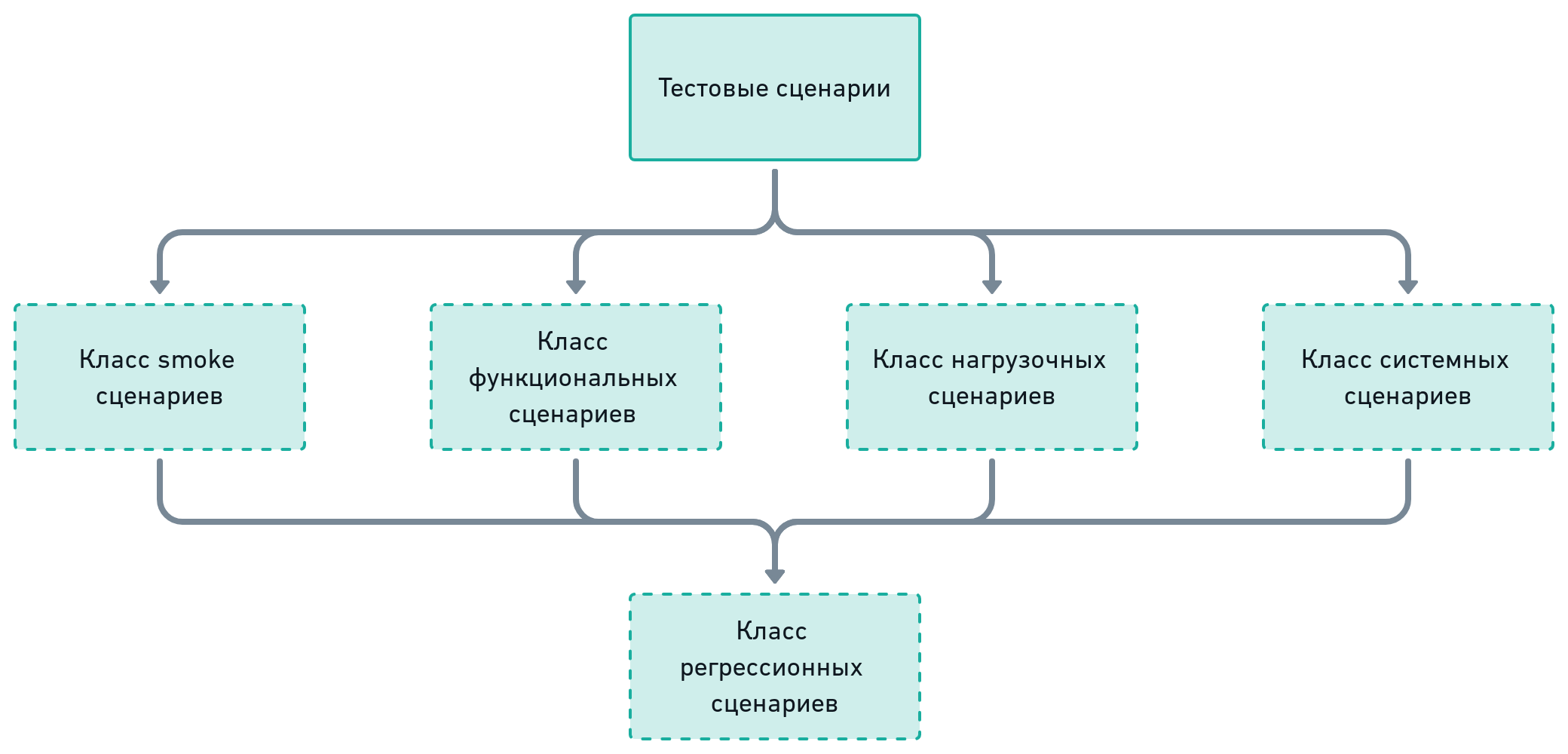


Рис. 11 – Классификация тестовых сценариев.

* Функциональный класс. Обеспечивает проверку заявленного функционала. Проверяется работа приложения согласно написанной технической документации.
* Smoke класс. Проверка приложения на работоспособность. Проверяется работа приложения независимо от бизнес-требований. Успешное smoke тестирование позволяет приступить к другим типам тестирования.
* Нагрузочный класс. Проверяется работа приложения под большой нагрузкой, что позволит увидеть слабые и сильные стороны конечного продукта.
* Системный класс. Проверяет функционал интеграции с другими приложениями или системами.
* Регрессионный класс. Подразумевает под собой множество различных тестов, которые прошли успешно в старой версии приложения. Позволяет легко выявлять ошибки, которые появляются в новых версиях приложения. Количество регрессионных тестов увеличивается с каждым новым релизом.

Структура сценариев каждого класса отличается подходом к тестированию, однако можно составить общую схему для всех классов.

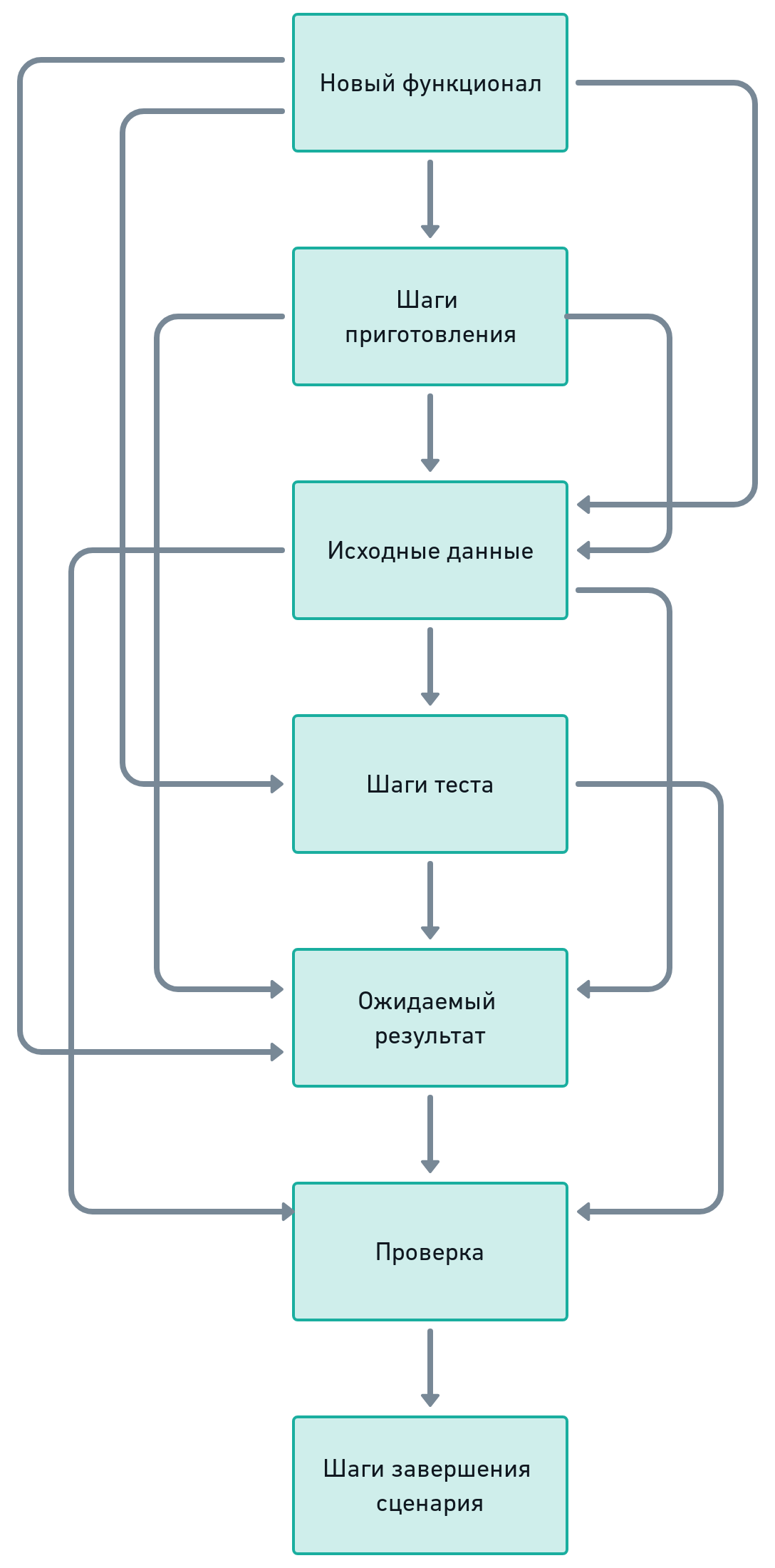


Рис. 12 – Общая схема формирования тестовых сценариев

Любой тестовый сценарий инициирован введением нового функционала в приложение. Далее написание сценария зависит от выбранного класса.

Сценарий может состоять из следующих блоков:

1. Шаги приготовления. Используется, если в сценарии необходимы какие-либо приготовления, например – установить соединение с базой данных, открыть браузер и подключиться к Apache Kafka. Необходимы, если множество тестов требуют одних и тех же исходных условий
2. Исходные данные. Используется, если в сценарии задействованы исходные данные, такие как список пользователей, платежей, покупок и т.д. Может быть один тестовый набор исходных данных для множества сценариев.
3. Шаги теста. Используются для формирования набора действий конкретного сценария. Обычно уникальны в рамках одного теста, однако могут быть частично переиспользованы в других сценариях.
4. Ожидаемый результат. Результат, который ожидается согласно техническому заданию. Может быть частью блока исходных данных.
5. Проверка. Получение результата непосредственно из приложения и сравнение его с ожидаемым результатом. Либо иная проверка работы приложения.
6. Шаги завершения сценария. Используется, если сценарий не сохраняет консистентность приложения.