МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«Автоматизація сайту засобами Python»**

Виконав: студент 2 курсу, групи 8.1210-з
спеціальності
121 інженерія програмного забезпечення
(шифр і назва спеціальності)
Н.В. Лабенська
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, к.т.н. Мухін В.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри фундаментальної математики, д.т.н. Гребенюк С.Н.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії,
к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2021 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТА

Лабенська Наталія Вікторівна
(прізвище, ім’я та по-батькові)

1. Тема роботи (проекту) Автоматизація сайту засобами Python

керівник роботи (проекту) Мухін В.В., доцент кафедри програмної інженерії, к.т.н.
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від «« 09 » 06 2021 року № 850-с

2. Строк подання студентом роботи

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
Презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	3.09.2021	
2.	Збір вихідних даних.	12.09.2021	
3.	Обробка методичних та теоретичних джерел.	23.09.2021	
4.	Розробка першого та другого розділу.	05.10.2021	
5.	Розробка третього розділу.	10.11.2021	
6.	Оформлення та нормоконтроль кваліфікаційної роботи.	19.11.2021	
7.	Захист кваліфікаційної роботи.	09.12.2021	

Студент _____ Н.В.Лабенська
(підпис) (ініціали та прізвище)

Керівник роботи _____ В.В.Мухін
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ С.П. Швидка
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Автоматизація сайту засобами Python»: 87 с., 29 рис., 35 джерел, 1 додаток.

PYTHON, ВИДИ ТЕСТУВАННЯ, MANUAL TESTING, AUTOMATION TESTING, STLC, TEST CASE.

Об’єкт дослідження – специфіка існуючих методів і засобів тестування програмного забезпечення.

Мета роботи: закріплення, розширення, систематизація знань в рамках досліджуваної предметної дисципліни, за рахунок проведення аналізу специфіки використання сучасних підходів і програмних засобів автоматизації тестування веб-сайтів.

Метод дослідження – аналіз, дослідження процесу автоматизації тестування на прикладі веб-сайтів.

SUMMARY

Master’s qualifying paper «Site Automation by Means of Python»: 87 pages, 29 figures, 35 references, 1 supplement.

PYTHON, TESTING TYPES, MANUAL TESTING, AUTOMATION TESTING, STLC, TEST CASE.

Object of the study – specifics of existing methods and means of software testing.

Aim of the study: consolidation, expansion, systematization of knowledge within the studied subject discipline, by analyzing the specifics of the use of modern approaches and software tools to automate testing of websites.

Methods of research – analysis, research of the process of automation of testing on the example of websites.

ЗМІСТ

Завдання на кваліфікаційну роботу 2
Реферат 4
Summary 5
Вступ 7
1 Аналіз особливостей тестування програмного забезпечення 9
1.1 Основні положення та принципи тестування 9
1.2 Огляд існуючих типів тестування 17
1.3 Тестування у життєвому циклу розробки сучасних програмних продуктів 24
2 Аналіз специфіки автоматизованого тестування програмного забезпечення 32
2.1 Актуальність автоматизації процесу тестування 32
2.2 Базові складові та етапи автоматизованого тестування 34
2.3 Переваги та недоліки автоматизованого тестування 42
3 Дослідження процесу автоматизації тестування на прикладі веб-сайтів 45
3.1 Аналіз існуючих аналогів на ринку автоматизації тестування 45
3.2 Обґрунтування вибору засобів розробки 52
3.3 Проектування та розробка системи автоматизованого тестування веб-сайтів 64
Висновки 80
Перелік посилань 82
Додаток А Приклад фрагменту реалізації системи 85

ВСТУП

Актуальність теми дослідження полягає у високому ступені затребуваності на ринку інформаційних технологій послуг з тестування програмного забезпечення (ПЗ). Вивчення можливостей організації і проведення процесу тестування сучасних складних програмних продуктів (ПП) дозволяє істотно знизити матеріальні і тимчасові витрати на підтримку проекту та програмного коду. Для забезпечення конкурентоспроможності сучасного ПЗ необхідно постійно підвищувати якість випущених ПП, що вимагає автоматизації рутинних процесів по перевірки роботи програм в різних умовах згідно з установленими при проектуванні специфікаціям. Все це обумовлює доцільність дослідження даної предметної області в рамках роботи.

Об'єкт дослідження: специфіка існуючих методів і засобів тестування програмного забезпечення.

Предмет дослідження: можливості використання технічних засобів для функціонального тестування веб-сайтів.

Мета роботи полягає в закріпленні, розширенні, узагальненні та систематизації знань в рамках досліджуваної предметної дисципліни, за рахунок проведення аналізу специфіки використання сучасних підходів і програмних засобів автоматизації тестування веб-сайтів.

Завдання дослідження:

– Аналіз основних положень та принципів тестування.

– Огляд існуючих типів тестування.

– Аналіз тестування у життєвому циклу розробки сучасних програмних продуктів.

– Аналіз специфіки автоматизованого тестування програмного забезпечення.

– Дослідження процесу автоматизації тестування на прикладі веб-сайтів.

В рамках першого розділу наведено результати аналізу особливостей тестування програмного забезпечення, основні положення та принципи тестування програмного забезпечення. Здійснено огляд існуючих типів тестування, надано їх класифікацію та прояснено їх відмінності. Описано місце тестування у життєвому циклу розробки сучасних програмних продуктів.

В рамках другого розділу наведено результати аналізу специфіки автоматизованого тестування програмного забезпечення. Визначено актуальність автоматизації процесу тестування, його базові складові та етапи, наведено основні переваги та недоліки використання та впровадження автоматизованого тестування.

В рамках третього розділу наведено результати дослідження процесу автоматизації тестування на прикладі веб-сайтів. Виконано аналіз існуючих аналогів на ринку автоматизації тестування, проведено обґрунтування вибору засобів розробки, здійснено проектування та розробку системи автоматизованого тестування веб-сайтів.

1 АНАЛІЗ ОСОБЛИВОСТЕЙ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Основні положення та принципи тестування

Інформаційні системи (ІС) як типове сучасне ПЗ з точки зору системного аналізу є складними системами, оскільки складаються з безлічі різних частин: функціональних модулів, серверів різного напрямку, баз даних, методів передачі даних. В спроектованих системах з безліччю взаємозв'язків між різними компонентами, ІС і інших взаємних інтеграцій підвищується шанс виходу з ладу одного або декількох модулів. Це може привести до виходу з ладу інших модулів, або повне руйнування всієї ІС та припинення її роботи. ІС повинна складатися та містить в собі методи обробки позапланових ситуацій, щоб бути більш стійкою до помилок і гарантувати відмовостійкість по технічнім умовам під час виконання роботи і виникнення передбачених і непередбачених помилок. Але обробка непередбачених ситуацій це лише один з безлічі методів, які підвищують надійність системи. Будь-яка ІС покривається тестуванням, будь це мануальне або автоматизоване тестування.

З цієї причини розробка тестових сценаріїв за різними методиками для ІС є актуальною протягом усього життєвого циклу розробки. Тестування програмного забезпечення (STLC) [1] - це систематичне тестування різними діями, яке проводитиметься в плановому порядку для підвищення якості продукту.

Тестування (software testing) - діяльність, яка виконується для оцінки та вдосконалення програмного забезпечення. Ця діяльність, в загальному випадку, базується на виявленні дефектів і проблем в ПЗ.

Тестування програмних систем складається з динамічної верифікації поведінки програм на кінцевому (обмеженому) наборі тестів, обраних

відповідним чином з зазвичай виконуваних дій прикладної області та забезпечують перевірку відповідності очікуваної поведінки системи [1].

Тестування проводиться відповідно до визначених цілей (можуть бути задані явно або неявно) і з різним рівнем точності. Визначення мети точним чином, що виражається кількісно, дозволяє забезпечити контроль результатів тестування.

Тестування ПЗ – виробнича діяльність, спрямована на визначення оцінки та забезпечення підвищення якості ПЗ. Необхідність подібної діяльності базується на важливості своєчасного виявлення різних проблем і дефектів в розроблюваних програмних системах.

Тест – виконувана тестова процедура з заданими вхідними даними, набором вихідних умов і варіантами очікуваних результатів, яка повинна бути виконана для заданої мети. В якості такої мети, як правило, виступає перевірка потрібного ПО або проведення верифікації роботи програми по конкретним вимогам.

Компонент – набір функцій, який буде використовуватися багаторазово в різних тестах [4].

Створення тестів або компонентів відбувається шляхом запису сеансу роботи з додатком або веб-сайтом. Також існує можливість ручного створення тестового скрипту, використовуючи в поле ключових слів ключові слова з попередньо створеного сховища об'єктів.

Щоб створити тест або компонент необхідно написати (записати) скрипт виконання даного тесту.

Існує два варіанти запису скрипту:

– автоматично записати сеанс роботи з додатком або сайтом;

– сформувати сховище і використовувати ці об'єкти для додавання кроків вручну або поле коду [7].

При тестуванні сайту або програми, можна параметризувати тест або компонент, щоб перевірити, як додаток виконує ті ж операції з іншими даними. В цьому випадку можна скористатися таблицею даних або

генератором випадкових чисел. Кожен запуск сеансу, який використовує параметризацію, називається ітерація. Також можна використовувати і вихідні величини, щоб витягувати дані з тесту або компоненту. Це дозволить використовувати витягнуті дані протягом виконання сеансу в інших частинах тесту або компонента.

Процес тестування ПО в загальному вигляді наведено на рис.1.1.

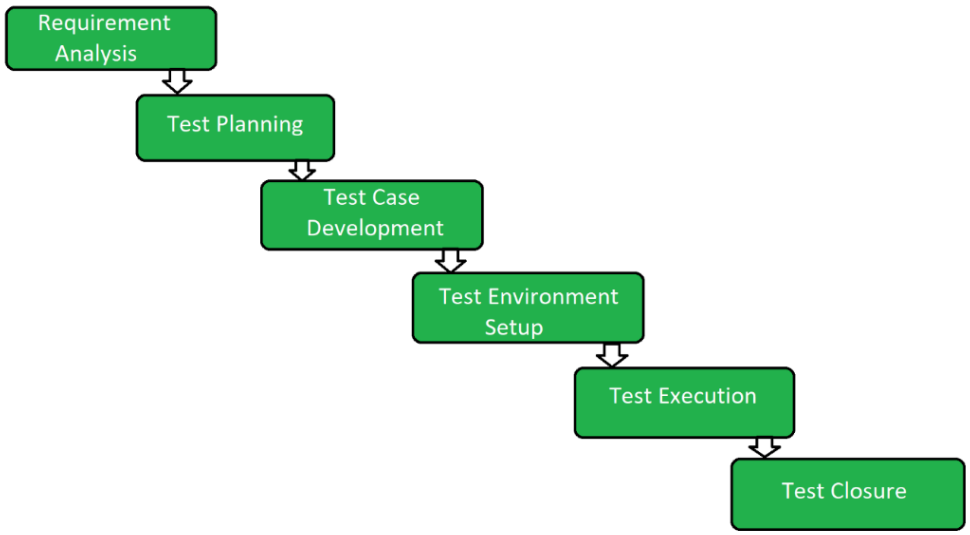

Рисунок 1.1 – Процес тестування ПЗ

Тестування ПЗ на практиці полягає в динамічній перевірці поведінкових моделей програм на заданому наборі тестових сценаріїв. Подібні моделі вибирають зі звичних наборів виконуваних дій в конкретній прикладній предметній області, що дозволяє забезпечити ефективну перевірку відповідності роботи системи очікуваній від неї поведінки.

Тестування ПЗ проводиться на базі чітких явних або не явних цілей, виконуючись з різним рівнем точності. Визначення мети за допомогою кількісних методів сприяє забезпеченню можливостей контролю підсумкових результатів [1].

Тестові сценарії (ТС) застосовують для перевірки різних функціональних вимог і оцінки не функціональних вимог. У ряді випадків

застосовуються тести, в яких кількісні параметри проведених результатів тільки в загальному вигляді задовольняють поставленим цілям тестування.

ТС дозволяє описати перевірку роботи ПЗ, яку здатна виконати будь-яка людина з виробничої команди, проте найчастіше це є роботою штатних або віддалених інженерів-тестувальників. Набір тестових сценаріїв часто називають тестовим набором (ТН).

ТС повинен допомогти провести перевірку продукту без ознайомлення з усією документацією. Написаний один раз, він зручний в підтримці, тому що економить багато часу і сил тестувальникам [9].

Для написання тестових сценаріїв, використовуються, як правило, такі елементи:

– Ідентифікатор - унікальний ідентифікатор тест-кейса. Його зручно використовувати для однакового розуміння, про яку перевірці йдеться.

– Назва - короткий опис суті перевірки. Повинно бути компактним і бути зрозумілим.

– Опис - опис елемента тестового сценарію.

– Передумова - дії, які потрібно виконати перед запуском тесту.

– Інструкція - це кроки тесту, які необхідно виконати для отримання очікуваного результату.

– Test Data - дані, які використовують тестові сценарії.

– Очікуваний результат - сама перевірка, що ми очікуємо отримати після виконання всіх кроків тестування [4].

В ЖЦ програмного продукту передбачено тестування ПЗ (STLC) наступними етапами зі своїми результатами і критеріями входу:

– Аналіз вимог – без специфікації неможливо правильно протестувати ІС, оскільки вони містять в собі перелік того, що треба продукту для поставленої мети проекту. Команда забезпечення якості (QA) ознайомлюється з цими вимогами, та приходять до розуміння, що повинні тестувати і як повинен ввести себе продукт. Вимоги можуть бути функціональні, які

описують поводження системи, або нефункціональні які описують експлуатаційну характеристику системи, наприклад вимоги до безпеки.

– Планування тестування – на цьому етапі QA планує будь використовувати методи і стратегії для тестування, оцінюючи витрати ресурсів і можливе покриття тестами. Зазвичай на цьому етапі створюється тест план, який описує обсяг робіт з тестування [2].

– Проектування тест-кейсів – процес проектування і створення тест-кейсів, відповідно до визначених раніше критеріями якості, цілями тестування, критеріями приймання.

– Налаштування тестового середовища – тестові кейси готуються до інтегрування в ЖЦ проекту, вбудовуються під необхідні вимоги.

– Виконання тесту – на цьому етапі QA починає виконання тестових кейсів, які були підготовлені на попередньому етапі. Спрямований на пошук помилок і можливих проблем ІС. Якщо один з тестових кейсів буде заблокований через дефект або іншої причини, такий тест кейс називається заблокованим. Такі тест-кейси повторно виконуються після усунення блокуючого фактору.

– Закриття тесту – на останньому етапі аналізується результат. Створюється звіт, документація, якщо необхідно – дефекти / баги, помилки за рівнем їх складності.

Сучасні моделі ЖЦ розробки ІС є ітераційні. При цьому довжина однієї ітерації може варіюватися в досить широкому діапазоні. Все залежить від швидкості змін вимог і частоти виходу релізів. Дотримуючись цього, ЖЦ тестування виражається замкнутої послідовністю дій з восьми етапів:

– стадія 1 – загальне планування й аналіз вимог – проаналізувавши вимоги можливе визначення об'єкта і мети тестування, оцінити тимчасові витрати, оцінити складності, визначити можливі блокери;

– стадія 2 – уточнення критеріїв приймання – формування і уточнення ознак і метрик можливостей або необхідність початку / тимчасове зупинення / відновлення тестування або його припинення;

– стадія 3 – уточнення стратегії тестування – уточнення планування на локальному рівні, які є актуальними на поточній ітерації ЖЦ;

– стадія 4 – розробка тест-кейсів, на даній стадії розробляється, уточняються, переглядаються, допрацьовуються і проводяться інші дії з тест-кейсами, наборами тест-кейсів, сценаріями та іншими артефактами, які будуть використовуватися при тестуванні;

– стадія 5 – виконання тест-кейсів – безпосередньо процес виконання тестових-кейсів, в процесі якого виявляються дефекти, які обробляються в наступній стадії;

– стадія 6 – фіксація знайдених дефектів – для формування розуміння послідовності відтворення дефекту, можливі причини виникнення, уточнення терміновості і важливості подальшої обробки даного дефекту;

– стадія 7 – аналіз результатів тестування – аналізується обсяг результатів, оцінка зусиль і плану ресурсів. Готуються дані для формування наступного етапу;

– стадія 8 – звітність – формування звітності по всім тест-кейсам й по всій ітерації циклу. Необхідно для визначення подальших обсягів завдань в наступній ітерації.

Кожна виділена стадія в сукупності являє собою черговий цикл ЖЦ тестування. Кожну ітерацію можна розглядати як перевірений функціонал або продукт, який відповідає вимогам за винятком виявлених дефектів, які будуть залишатися в наступних ітераціях, до моменту їх виправлення. Ці характеристики визначаються і деталізуються на основі технічного завдання і вимог до ПЗ. В процесі проектування ПЗ вони конкретизуються і деталізуються, у свою чергу ІС повинна відповідати і бути наближеною до еталонного зразка на кожній стадії тестування [3].

Тестування ПЗ охоплює всі основні стадії ЖЦ ІС, паралельний циклу процесів розробки: постановка задачі, проектування, написання тест-кейсів, перевірка тестів, виконання тестів і створення звітності по тестах. Щоб орієнтуватися в цьому, варто розглянути два основні підходи.

Підхід орієнтований на вивчення бізнес-логіки програмного забезпечення ІС при проектуванні тестових сценаріїв. Тут головним завданням є покриття кожної гілки алгоритму хоча б один раз. При такому підході специфікації відходять на другий план, але все одно є важливою частиною [4].

При другому підході враховується один з найпоширеніших принципів: результат повинен перевищувати трудовитрати. Результат вимірюється в можливості виявити дефект при виконанні тестового сценарію. Більш корисні виявленні дефекти ті, що є серйозними і критичними для системи. Трудовитрати вимірюються в кількості робочого часу, необхідного для виконання тесту, складання звітності та перевірки результатів. При цьому у кожного тест-кейса повинен бути чіткий план виконання, вхідних і вихідних даних.

Якщо врахувати, що в складі ІС може використовуватися не тільки програмні компоненти, але й апаратні, то і у звіті її випробувань повинні бути зображені показники обраних серверів, робочих таргетів, ефективність розробленого підходу експлуатації системи, а також мережевого обладнання, а саме їх продуктивність і надійність. Виникає необхідність проводити у відповідність всім вимогам, що внесені в проект, комплексного тестування. Таке тестування ІС складається з етапів:

1. Детальне вивчення вимог проекту, самої системи і її супровідних документів.

2. Підбір та інтеграція автоматизованої системи відстежування помилок (bug tracking).

3. Робота з тестування ІС, яка складаються з:

– тестування всіх модулів ІС на коректну роботу;

– тестування максимального навантаження ІС та її продуктивності;

– перевірка модулів об'єднуючи їх в групи, перевіряючи безпосередньо їх взаємозв'язок;

– тестування на відмови: вихід з ладу ІС, або її модулів, несподівана робота модуля, людський фактор та ін.;

– загальна перевірка системи;

– перевірка внутрішніх зв’язків;

– регресійні тестування – перевірка на те, що виправлені дефекти знов не зламаються при інших змінах.

4. Тестування готового продукту до релізу.

5. Аналіз створених звітів і покращення стратегії до виявлення нових помилок.

Комплексне тестування обернено на пошук некоректності системи її вихідними цілям, а не для тестування функцій остаточно зібраної ІС. Тому, в комплексному тестуванні бере участь ІС, а саме опис її вихідних цілей, вимоги та вся інша документація, яка буде поставлятися з системою. З такими цілями використовується тестування чорної коробки [5].

Методологія тестування припускає цілі стратегії та підходи до тестування ІС для гарантування того, що продукт відповідає технічним завданням і готовий до експлуатації.

Методики тестування включають тестування того, що ІС працює відповідно до вимог, та не має небажаних сторонніх ефектів при використанні способами, що виходять за межі проектних параметрів, і в гіршому випадку буде відмовостійкою.

Для того, щоб ІС була протестована в більшому обсязі необхідно використовувати різні підходи й способи методології тестування. Методології тестування охоплюють багато що: від модульного тестування окремих функцій, інтеграційного тестування різних модулів, до спеціалізованих форм тестування, які є такими як продуктивність та безпека.

Оскільки ПЗ стають більш складнішими та взаємозв'язаними, а також з великою кількістю різних пристроїв і платформ, які необхідно протестувати, важливо мати надійну методологію тестування, щоб переконатися, що розроблене ПЗ було повністю протестоване, що воно відповідає зазначеним

вимогам і може успішно працювати у всіх очікуваних середовищах з необхідним рівнем зручності використання і безпекою.

1.2 Огляд існуючих типів тестування

Сучасні види тестування ПЗ, в залежності від необхідних поставлених цілей і технічних завдань, часто класифікують на такі узагальнені групи [5]:

- Функціональний вид тестування ПЗ.
- Нефункціональний вид тестування ПЗ.
- Вид тестування ПЗ, пов'язаний з можливими змінами.

Функціональні тести ґрунтуються на наборах функцій і їх особливостях, а також враховують специфіку взаємодії з іншими системами або підсистемами, як правило, представленими на таких рівнях тестування: модульному (компонентному), інтеграційному, системному і приймальному. Такі види тестування призначені для розгляду зовнішніх моделей поведінки розроблюваних систем.

Тестування ПЗ різними авторами часто класифікується [1-8,10,11]:

- за рівнем використання ПЗ;
- за загальним знання системи;
- за досліджуваним об'єктом тестування;
- за рівнем ізольованості складових компонентів;
- за ознакою позитивності виконуваних сценаріїв;
- за часом здійснення тестування;
- за рівнем підготовленості до процесу тестування;
- по загальному рівню тестування.

До найбільш поширених видів функціональних тестів найчастіше ставляться такі типи тестування [7]:

- функціональне.

– безпеки доступу.

– специфіки взаємодії.

Більш повна класифікація існуючих видів тестування ПЗ наведена на рис.1.2.

Рисунок 1.2 – Класифікація існуючих видів тестування ПЗ

Нефункціональне тестування призначене для здійснення опису тестів, які є необхідними для ідентифікації різних наборів характеристик ПЗ, вимірюваних різними величинами. Даний вид тестування відображає модель специфіки роботи системи. Існують такі основні типи не функціональних тестів:

1. Тестування рівня продуктивності ПЗ:

– навантажувальний;

– стресовий;

– стабільності.

2. Тестування установки.

3. Тестування зручності (комфорту) використання.

4. Тестування на відмовостійкість.

5. Конфігураційне тестування [4].

За рівнем використання ПЗ при тестуванні бувають:

– статичну (static);

– динамічний (dynamic).

Статичне тестування являє собою процес комплексного аналізу етапів розробки ПЗ без запуску самої програми. Даний тип тестування передбачає перевірку працездатності коду, специфікацій вимог до програмного продукту, архітектурі, дизайну, інтерфейсу і т.д.

Динамічне тестування являє собою діяльність, яка передбачає процеси експлуатації ПЗ в різних режимах. Даний тип тестування передбачає виконання запуску програми, перевірки всіх функціональних модулів програми та зіставлення фактичного поведінки ПЗ з очікуваним.

За загальним знання системи спеціалістом з тестування виділяють:

– тестування без урахування внутрішньої будови (метод «чорного ящика»);

– тестування з урахуванням внутрішньої будови і доступом до коду (метод «білого ящика»);

– тестування з частковим доступом (метод «сірого ящика»).

При тестуванні без урахування будови програми ідеї і операції тестування ПЗ формуються на основі висування припущень про можливі сценарії, які реалізуються користувачами в першу чергу. У зв'язку з цим метод «чорного ящика» в літературі часто позначають як «поведінкове тестування». Прикладом подібного типу тестування ПЗ є функціональне тестування, в такому випадку фахівець запускає програмний додаток на виконання і в процесі його запуску тестує основні функціональності програми, використовуючи в своїй роботі наявний призначений для користувача інтерфейс для введення вхідних даних та перевірки результатів.

Метод з урахуванням внутрішньої будови часто називають «скляним ящиком» або «відкритим ящиком». При тестуванні за даним методом тестувальник засновує ідеї і процедури тестування ПЗ на знанні складу і внутрішньої логіки програми, при цьому зразки передбачуваної поведінки користувачів не важливі. У практиці розробки подібне тестування проводиться або розробниками, що створили код, або їх колегами, які володіють основами і розумінням використаних мов програмування. Типовим видом подібного тестування є модульне.

Метод «сірого ящика» являє собою суміш підходів, що використовуються при тестуванні «чорного ящика» і «білого ящика». При цьому тестування, по суті, орієнтоване на кінцевого користувача, тобто застосовуються сценарії поведінки користувача. Однак, тестувальник має загальне або часткове знання про те, яким чином влаштована частина аналізованого ПЗ [15].

По об'єкту тестування ПЗ розрізняють наступні види [18]:

– функціональне;
– тестування користувальницького інтерфейсу програми (UI);
– тестування коректності локалізації та інтернаціоналізації;
– тестування загальної продуктивності програми;
– тестування безпеки застосування програми на практиці;
– перевірки зручності;
– тестування сумісності програми з зовнішніми модулями або стороннім ПЗ.

Функціональне тестування здійснюється для послідовного виявлення існуючих помилок в роботі програми за допомогою зіставлення фактичного і очікуваного результатів використання окремих можливостей ПЗ. При цьому, для отримання підсумкового фактичного результату з метою верифікації сценарію програму необхідно виконати, найчастіше в ручному режимі. Головним джерелом при формулюванні очікуваного результату є набори специфікацій по функціональним вимогам до ПЗ.

Тестування інтерфейсу ПЗ являє собою популярний в останні роки тип тестування програм, націлений на перевірку і верифікацію коректності використання створених елементів інтерфейсу користувача.

Тестування інтернаціоналізації - вид тестування, при якому перевіряється ступінь готовності програмного додатка до роботи з різними налаштуваннями мови. Це може бути перевірка коректності відображення шрифтів в пунктах навігаційного меню, результатів відображення тексту в разі виконання пошуку або сортування даних, здатність системи здійснювати обробку файлів, які містять імена на різних мовах [5].

Локалізаційне тестування здійснює перевірку ступеня коректності адаптації програмного продукту до роботи на заданій лінгвістичній мові, тобто, виконує перевірку якості перекладу термінів і фраз, логіки побудови програмного інтерфейсу і обробки даних [11].

Тестування продуктивності здійснюється з метою виявлення швидкодії роботи програми або її окремих частин при впливі на ПЗ певного, чітко заданого, навантаження. На практиці також виділяють навантажувальний і стресовий типи тестування ПЗ.

Тестування навантаження ПЗ дозволяє здійснити перевірку здатності програмного додатка функціонувати коректним чином при запланованому рівні навантаження. Стресове тестування перевіряє роботу програмного додатку в умовах, які поза обумовленою документацією. Призначенням даного виду тестування є ідентифікація найбільш слабкого місця в програмі, в тому числі в коді, базі даних, в обміні даними між спеціалізованими апаратними компонентами.

Тестування безпеки ПЗ здійснюється для проведення оцінки рівня вразливості програм до різних шкідливих дій на її структуру на рівнях локального додатку, операційної системи, комп'ютерної мережі.

Тестування зручності використання ПЗ здійснюється з метою виявлення рівня органічності застосування розробленого інтерфейсу користувачами, фактично дозволяє оцінити його інтуїтивність сприйняття.

Тестування сумісності є типом тестування програм, ключовою метою якого є виконання перевірки взаємодія створеного ПЗ з іншими зовнішніми програмами (браузерами, антивірусними програмами та ін.). Тестування сумісності роботи програми в різних браузерах часто називається крос-браузерним тестуванням. Тестування ПЗ під управлінням різних операційних систем часто називають як кросплатформенне тестування.

За рівнем ізольованості тестування компонентів програм виділяють [14]:

– інтеграційне, яке являє собою тестування взаємодії кількох компонентів одноразово;

– компонентне, що полягає в перевірці роботи логічних компонентів ПЗ;

– системне, необхідне для перевірки роботи всієї системи в зборі.

За критерієм «позитивності» тестових сценаріїв тестування класифікують на [16]:

– позитивне, проводиться за передбаченими тестувальником сценаріями, які передбачають штатну роботу програми в свідомо коректних умовах;

– негативне, використовуються тестові сценарії, які здійснюють перевірку ситуацій, пов'язаних з усіма можливими дефектами в системі. Найпростішим прикладом є перевірка роботи програми в разі введення недійсних або не правильних даних.

При наявності позитивних і негативних сценаріїв тестування пріоритетом користуються позитивні сценарії.

За часом проведення тестування виділяють [3]:

– тестування до передачі ПЗ кінцевому користувачеві (також називається альфа-тестуванням);

– тестування після передачі ПЗ кінцевому користувачеві (в літературі згадується як бета-тестування).

На практиці, як правило, в якості альфа-тестувальників запрошуються співробітники компанії розробника, наприклад: програмісти, тестувальники, системні адміністратори, дизайнери та ін.

Бета-тестувальники представляють собою обмежену і відібрану за заданими критеріями групу користувачів з цільової аудиторії, які отримують доступ до нової системи до того моменту, коли система стане доступна всім її користувачам.

За рівнем тестування виділяють [7]:

- тестування приймання ПЗ;
- тестування створеної функціональності ПЗ;
- регресійне тестування ПЗ;
- тестування оцінки значення критичного шляху;
- розширене тестування;
- тестування здачі ПЗ для сертифікації.

В наслідок проведених різних змін, наприклад, виправлення дефектів різних видів, ПЗ повинно пройти повторний цикл тестування для підтвердження правильного програмного рішення щодо усунення існуючих проблем. Існують такі підвиди тестування, що є необхідними для проведення після установки з метою підтвердження працездатності програмних систем, зокрема [1]:

- Димове.
- Регресійне.
- Тестування збірки.
- Верифікація справності.

1.3 Тестування у життєвому циклу розробки сучасних програмних продуктів

Підсумковими даними і вищим пріоритетом в процесі вибору ключових показників якості ПЗ є прикладне призначення, функції та придатність для вирішення різних завдань. Опис перерахованих властивостей є базою для визначення значень інших, менш значущих, характеристик і показників якості ПЗ, що є пріоритетними у ЖЦ розробки.

Технічні можливості та ступінь точності вимірювання чисельних значень таких атрибутів і характеристик часто сильно обмежені в силу специфіки їх змісту.

Це багато в чому визначає ефективні діапазони значень кожного з критеріїв, які обираються строком на основі практичного досвіду або шляхом аналізу наявних прецедентів в специфікаціях вимог різних проектів.

Всі існуючі процеси вибору метрик і шкал, що використовуються для формалізації чисельних характеристик якості ПЗ слід класифікувати за двома етапами [19]:

– вибір наборів вихідних даних, які адекватно відображають ключові особливості ЖЦ проекту ПЗ і специфіку роботи з продуктом його споживачів, що впливають на різні характеристики якості;

– вибір конкретних шкал і метрик обчислення характеристик і показників якості проекту для їх подальшої оцінки і зіставлення відповідних специфікацій в процесі проведення кваліфікаційних випробувань або здійснення сертифікації на заданих етапах ЖЦ ПО.

Оцінці характеристик якості розробленого ПЗ і використовуваних компонентів на різних етапах ЖЦ присвячений офіційний міжнародний стандарт ISO 14598, який включає до свого складу шість частин. Відповідно до цього застосовується така схема оцінки характеристик і показників якості програм [2]:

– установка вихідних вимог, що включає в себе формування цілей випробувань, визначення конкретних типів метрик оцінки ПЗ, оцінка адекватності показників і значень виділених атрибутів якості програмного продукту;

– вибір найбільш ефективних метрик оцінки якості ПЗ, з встановленням системи рейтингу пріоритетів за наявними характеристиками і атрибутам, після чого відбувається виділення необхідних критеріїв для здійснення експертиз і вимірювань;

– проектування і планування ключових процесів оцінки підсумкових характеристик і показників якості в ЖЦ ПЗ;

– здійснення вимірювань для отримання підсумкових оцінок, після чого виконується порівняння отриманих результатів з наявними критеріями і вимогами, на базі чого здійснюється узагальнення і оцінка підсумкових результатів.

Місце тестування у ЖЦ розробки програм наведено на рис.1.3.

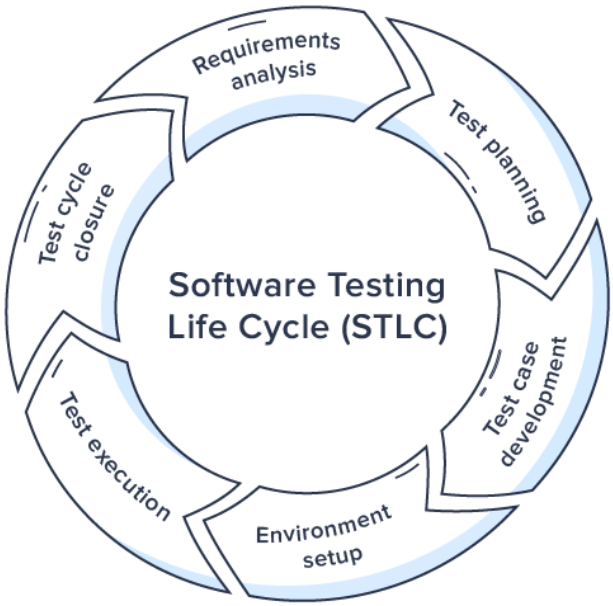

Рисунок 1.3 – Місце тестування у ЖЦ розробки програм

В контексті цього питання необхідно формалізувати ряд ключових термінів, які широко застосовуються при проведенні оцінки якості ПЗ.

Функціональні можливості полягають у здатності ПЗ забезпечувати вирішення поставлених завдань, які в повній мірі задовольняють наявним потребам замовників і кінцевих користувачів при використанні розробленого програмного продукту в конкретних умовах.

Функціональна придатність ПЗ полягає в описі характеристики і атрибутів, здатних визначити призначення, основні функції програмного засобу, його номенклатуру, ступінь відповідності створеним технічним завданням і вимогам замовника та кінцевого користувача ПЗ [3].

Коректність ПЗ - це здатність забезпечувати прийнятні для користувача підсумкові результати і ефекти зовнішнього інтерфейсу продукту.

Здатність до взаємодії ПЗ являє собою деяку властивість компонентів і програмних засобів, що полягає в ефективній взаємодії з іншими компонентами, що знаходяться у внутрішньому або зовнішньому середовищі [9].

Захищеність ПЗ полягає в здатності використовуваних компонентів ПЗ забезпечувати достатній рівень захисту програми від різних негативних впливів.

Надійність ПЗ являє собою ступінь забезпечення програмою мінімальної ймовірності відмови в процесі використання ПЗ в режимі реального часу.

Ефективність ПЗ - це властивість, що забезпечує необхідний рівень продуктивності рішення функціональних завдань, на базі оцінки кількості застосовуваних обчислювальних ресурсів в заданих умовах [10].

Практичність ПЗ - це властивості, що описують складності при його вивченні і використанні на практиці, а також доцільність використання кваліфікованими користувачами.

Супроводжуємість ПЗ - ступінь підтримки можливостей по модифікації, вдосконаленню і зміні конфігураційних налаштувань та функціональних можливостей [11].

Портативність ПЗ - рівень підготовленості до можливого переносу в сторонні апаратно-операційні системи і середовища [15].

Оцінка рівня коректності ПЗ як правило полягає в формальному визначенні показника відповідності програмного продукту вихідним вимогам контракту між виконавцем і замовником, технічного завдання і сформульованих специфікацій на ПЗ і його окремі компоненти або модулі. Шляхом проведення верифікації визначається рівень відповідності всіх складових ПЗ вихідним вимогам (аж до програмних модулів, коду, коментарів по опису інформації).

Оцінка здатності до взаємодії полягає в ідентифікації якості спільної роботи всіх наявних компонентів ПЗ і баз даних (БД) з іншими наявними в експлуатації прикладними системами або компонентами відмінних операційних платформ, з урахуванням порядку взаємодії з кінцевими користувачами в стилі, що є максимально зручним для переходу до інших обчислювальних систем.

Оцінка ступеня захищеності ПЗ включає формалізацію повноти використання ефективних і сучасних методів і засобів захисту програмного продукту від можливих загроз і розрахунок досягнутого рівня безпеки функціонування ПЗ [10].

Оцінка надійності ПЗ часто зводиться до вимірювання ряду кількісних метрик у використанні програмного продукту, зокрема такими метриками можуть виступати: завершеність, стійкість до дефектів і доступність до роботи.

Потреби в обчислювальних ресурсах в процесі вирішення конкретних завдань часто змінюються в залежності від основного складу і обсягів початкових даних. Для коректного визначення граничної здатності роботи окремої ІС зі створеним ПЗ необхідно виявити екстремальні і нормальні значення тривалості виконання програми.

Оцінка практичності ПЗ здійснюється експертами, маючи на увазі визначення зрозумілості, зручності використання і привабливості

програмного продукту. Дана оцінка є багато в чому суб'єктивною і не завжди може бути виражена в точних балах, при цьому ряд показників можна оцінити кількісно за ступенем трудомісткості і швидкості виконання обчислювальних операцій при використанні ПЗ, а також за обсягом складеної технічної та довідкової документації [16].

Супроводжуємість ПЗ можна оцінити шляхом визначення повноти і ступеня достовірності наявної документації про стан програмного продукту, всі зміни, для встановлення поточного стану версії ПЗ.

Оцінка мобільності ПЗ - являє собою якісне визначення кваліфікованими і досвідченими експертами ступеня адаптації, простоти розгортання, рівня сумісності програми з іншими рішеннями, яке виражається в балах. Чисельним чином дану характеристику ПЗ і загальну сукупність всіх його атрибутів доцільно оцінити в різних економічних умовах: трудомісткості, вартості, тривалості реалізації портування на різні платформи [19].

Якість розробленого ПЗ розумно оцінювати на базі критеріїв і метрик, які повинні [10]:

– кількісно характеризувати цільову функцію ПЗ;

– організувати можливості обчислення витрат, які необхідні для забезпечення заданого рівня якості, а також рівня впливу на показник якості програмного продукту різних зовнішніх факторів і явищ;

– бути прості у використанні і володіти мінімальною дисперсією.

В даний час на практиці для вимірювання характеристик якості ПЗ у ЖЦ розробки ІС застосовуються різні метрики.

Метрика якості (МЯ) ПЗ - це система визначення рівня якості програмного продукту. Вимірювання можуть бути глобальними (на рівні критеріїв) або локальними (на рівні окремих характеристик) [11].

У першому випадку використана система вимірювань дозволяє проводити порівняння різного ПЗ, проте самі вимірювання не можуть бути повноцінно проведені без урахування суб'єктивних людських оцінок основних властивостей програми.

У другому випадку вимірювання зазначених характеристик можуть бути виконані об'єктивно, проте інтегральна оцінка якості ПЗ все ж буде пов'язана з суб'єктивною інтерпретацією підсумкових оцінок.

Існує три види МЯ [17]:

– метрики ПЗ, що застосовуються для вимірювання його характеристик;

– метрики процесу, що застосовуються для вимірювання властивості самого процесу розробки програмного продукту;

– метрики використання ПЗ.

Перший тип МЯ включає до свого складу [7]:

– зовнішні метрики, які позначають властивості продукту, які безпосередньо бачить користувач;

– внутрішні метрики, які позначають властивості доступні тільки розробникам.

Зовнішні метрики позначають такі аспекти [8,12]:

– надійність ПЗ, що використовується для розрахунку можливого числа дефектів;

– функціональні можливості ПЗ для перевірки наявності та коректності реалізації всіх функцій розробниками;

– супроводу ПЗ для вимірювання всіх наявних ресурсів програмного продукту (обчислювальна швидкість, обсяг займаної пам'яті);

– застосування ПЗ для визначення рівня доступності для використання і вивчення в процесі експлуатації;

– вартості ПЗ для зведеної оцінки підсумкової ціни створеного програмного продукту.

Внутрішні метрики включають в свій склад [10]:

– методи визначення розміру ПЗ, використовувані за допомогою оцінки його внутрішніх характеристик;

– методи визначення складності ПЗ, необхідні для оцінки складності програмного продукту;

– метрики стилю ПЗ, використовувані для оцінки методів і технологій створення програмних компонентів продукту і супутніх документів.

Внутрішні метрики уможливлюють визначення продуктивності програмного продукту і є повністю релевантними зовнішнім.

Як зовнішні, так і внутрішні метрики задаються вже на початковому етапі формування переліку вимог до програмного забезпечення, що дозволяє використовувати їх в якості предмета управління досягненням необхідного рівня якості підсумкового програмного продукту [1].

Метрики ПЗ описуються комплексом різних типів моделей для оцінки різних властивостей або прогнозування. Всі необхідні вимірювання відбуваються після проведення калібрування всіх метрик ще на ранніх етапах створення проекту. Загальною мірою при цьому виступає ступінь трасуванню, яка визначається числом трас, які простежуються за допомогою створених моделей сценаріїв на базі діаграм UML і оцінкою загальної кількості [11]:

– функціональних вимог;

– сценаріїв роботи з програмою і дійових осіб;

– об'єктів, які включені в кожен сценарій;

– локалізацій вимог по кожному окремому сценарієм взаємодії користувача і програми;

– параметрів об'єкта.

Стандарт ISO / IEC 9126-2 під час ЖЦ розробки ПЗ часто використовується для визначення таких головних заходів [16]:

– розміру ПЗ в різних одиницях вимірювання, таких як кількість функцій, довжина рядків коду, розмір займаної постійної пам'яті;

– часу роботи системи і виконання кожного компонента;

– зусиль, тобто трудомісткості виконання операцій, загальної продуктивності праці;

– обліку кількості наявних помилок, відповідей системи, відмов у роботі.

Спеціальної мірою є рівень повторного використання компонентів, який визначається у вигляді відношення підсумкового розміру програмного продукту до розміру всієї розробленої системи в цілому. Прикладами метрик, що використовують наведені заходи є [7]:

– метрики оцінки підсумкового кількості об'єктів і числа повторно використовуваних;

– метрики оцінки загальної кількості наявних операцій, а також повторно використовуваних і тих, що генеруються в процесі;

– метрики оцінки кількості програмних класів, які успадковують зазначені специфічних операції;

– метрики оцінки числа класів, що впливають на залежність від них іншого класу;

– метрики оцінки числа користувачів конкретного класу.

При оцінці підсумкової кількості ключових величин досить часто застосовують середньостатистичні метрики, наприклад оцінка середньої кількості операцій в класах, середнє число спадкоємців або операцій класів і ін.

Використовувані заходи найчастіше є суб'єктивними і багато в чому залежать від отриманих знань експертів, які здійснюють кількісні оцінки зазначених атрибутів всіх компонентів аналізованого ПЗ [15].

На базі подібних атрибутів стає можливим обчислення часу розробки коду, рівня структурованості і якості програмного продукту, ступеня використання ключових можливостей і аспектів мови програмування, ступінь абстракції [14].

У якості метрик процесу найчастіше використовуються показники часу розробки, кількості помилок, які були знайдені на етапі проведення тестування ПЗ.

Таким чином, отримані під час написання даної глави результати аналізів будуть застосовані під час проведення подальших досліджень с автоматизації процесу тестування програмних застосувань.

2 АНАЛІЗ СПЕЦИФІКИ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Актуальність автоматизації процесу тестування

Якість сучасного ПЗ в першу чергу визначається відповідністю всіх очікувань зацікавлених у ньому сторін (замовник, користувачі, спонсори, розробники і тестувальники, фахівці з технічної підтримки, маркетологи та ін.). На практиці між усіма учасниками при створенні проекту виникають суперечності в думках про ПЗ і його якості. У зв'язку з цим завдання забезпечення якості програмного продукту обумовлює необхідність ідентифікації інтересів всіх основних осіб, залучених до процесу розробки, висунутих ними вимог і критеріїв. На основі врахування цієї інформації стає можливим знаходження оптимального рішення, що задовольняє всіх ключових критеріїв. Тестування ПЗ в цьому випадку є інструментом виявлення можливих невідповідностей між необхідним і підсумковим програмним продуктом [15].

Сучасних менеджерів і розробників ПЗ умови ринку часто змушують здійснювати створення програмних додатків з мінімальними витратами ресурсів в стислі терміни, що обумовлює підвищення частоти випадків зриву дати релізів продуктів. У зв'язку з цим програмістам досить часто доводиться видаляти в процесі розробки ПЗ одну або кілька ключових функціональностей для виконання вимог щодо термінів [16]. Зниження витрат в таких випадках можливо шляхом впровадження методів модернізації бізнес-процесів і автоматизованих засобів тестування ПЗ.

Автоматизоване тестування ПЗ являє собою процес управління всіма роботами по перевірці функціонування програмного додатка без необхідності ручного перевірки кожного сценарію тестувальником. Автоматизація робіт з проведення тестування ПЗ є особливо ефективною і необхідною для проектів,

в яких тестові скрипти багаторазово повторюються, набори тестових процедур регулярно запускаються на виконання. Подібне тестування ПЗ на всіх стадіях розробки та інтеграції проекту забезпечує значне прискорення процесів верифікації та валідації роботи програми [17].

Схему основних етапів автоматизації тестування ПЗ наведено на рис.2.1.

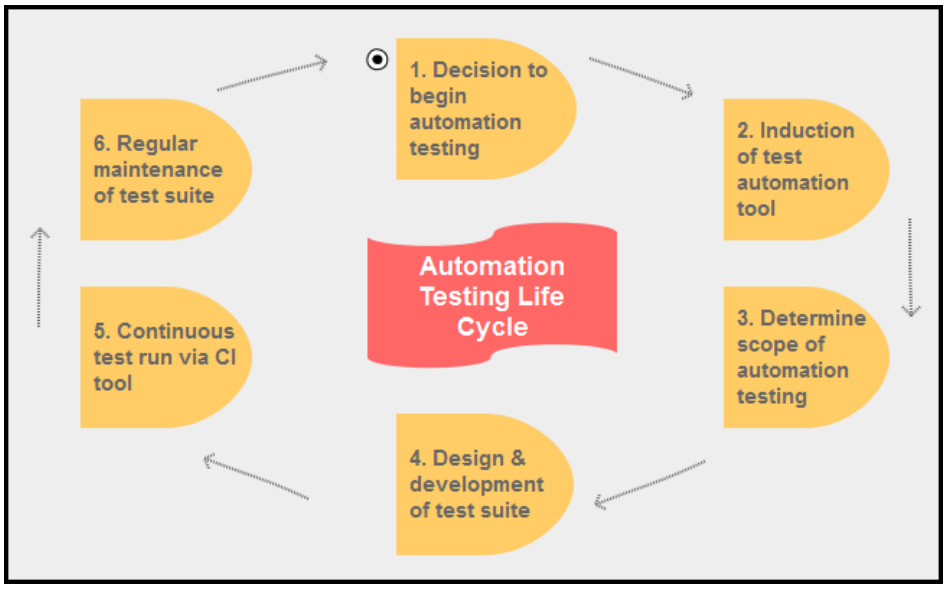

Рисунок 2.1 – Схема основних етапів автоматизації тестування ПЗ

Автоматизація процесів тестування обумовлена і може застосовуватися для невеликих або великих проектів. У зв'язку з тим, що в масштабних і активно розвиваються програмних додатках і проектах число тестів може перевищувати десятки і сотні тисяч, найбільш ефективним є застосування засобів і методів автоматизації регресійного типу тестування.

Це пов'язано з тим, що регресивні тести необхідні для перевірки функції, які містяться у зміненій системі, на адекватність, стабільність і коректність роботи. У подібній ситуації впровадження підходів до автоматизації процесів тестування ПЗ істотно скоротить обсяг робіт [3].

Автоматизація тестування ПЗ дозволяє зменшити витрати часу, який виділяється в компанії на тестування, а також спрощує сам процес, використовуючи спеціальні програмні засоби для виконання тестів і перевірки отриманих результатів тестування. Найбільш широко використовується

формою забезпечення автоматизації тестування є використання графічного інтерфейсу користувача.

В даний час на ринку ПЗ представлена велика кількість інструментів для автоматизації процесів тестування, зокрема, вони призначені для: тестування продуктивності програм, пошуку можливих витоків пам'яті при їх роботі, тестування інтерфейсу користувача.

Автоматизація корисна в наступних випадках:

– Першочергове завдання - заощадити час проектної команди.

– Тести повинні виконуватися для кожної збірки додатку.

– Проект тривалий або комплексний (складається з різних ітерацій).

– На виконання тест-кейсів витрачається багато часу і ресурсів.

– Проводиться навантажувальний або стрес-тестування.

– Потрібно скоротити поточний обсяг тестування з метою встигнути до певного строку.

Автоматизація може не бути корисною в наступних випадках [19]:

– Для виконання тестування потрібен людський інтелект і інтуїція.

– Процес тестування обмежений інтуїтивними або дослідницькими перевірками.

– Вимоги, що ставляться до існуючої функціональності, часто змінюються.

– Потрібно провести тестування лише одного разу.

2.2 Базові складові та етапи автоматизованого тестування

Процес автоматизації тестування - це інтелектуальна творчість IT-фахівців високої кваліфікації, але для досягнення поставлених цілей його теж необхідно вести планомірно. На кожному етапі наші фахівці вибирають правильну стратегію випробувань при перевірці якості досліджуваного об'єкта.

Загальний вигляд операцій при автоматизації тестування наведено на рис.2.2.

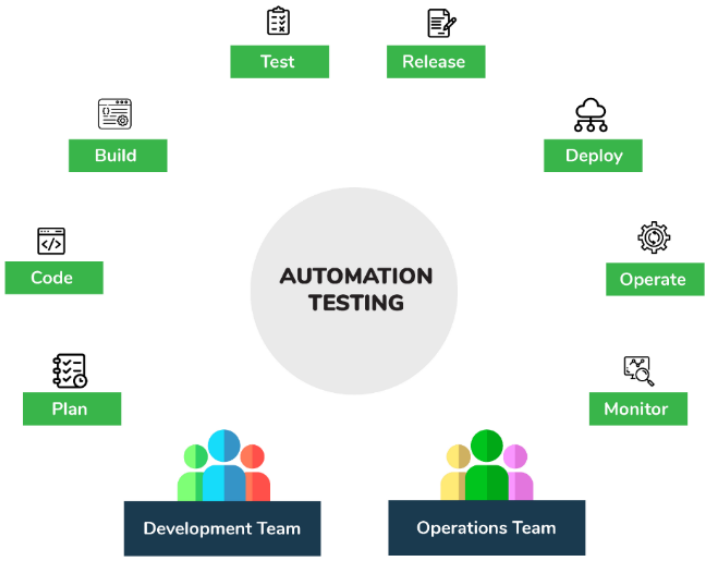

Рисунок 2.2 – Загальний вигляд операцій при автоматизації тестування

Фреймворк у тестуванні – набір інструкцій з автоматизації, які [20]:

– підтримують послідовність тестування;

– покращують структурування тесту;

– дозволяють використовувати мінімальну кількість коду;

– зменшують витрати на обслуговування коду;

– підвищують зручність повторного використання;

– дають можливість нетехнічних тестувальникам брати участь в кодуванні тестів;

– допомагають скоротити термін навчання використанню інструменту;

– включають дані всюди, де це необхідно.

Для автоматизації тестування програмного забезпечення використовують чотири типи фреймворків:

– платформа автоматизації на основі даних;

– фреймворк автоматизації на основі ключових слів;

– модульна платформа автоматизації;

– гібридна середу автоматизації.

Обсяг автоматизації - це область тестованої програми, яка буде автоматизована. Його допомагають визначити:

– Функції, важливі для бізнесу.

– Сценарії з великим об'ємом даних.

– Загальні функції додатків.

– Технічна здійсненність.

– Частота повторного використання бізнес-компонентів.

– Складність тестових випадків.

– Можливість використовувати одні і ті ж тестові сценарії для крос-браузерного тестування.

На етапі виконання тесту виконуються сценарії автоматизації. Сценаріями необхідно ввести тестові дані, перш ніж вони будуть запущені. Після виконання вони надають докладні звіти про випробування.

Виконання може бути виконано з використанням інструменту автоматизації безпосередньо або за допомогою інструменту управління тестуванням, який здійснить виклик інструмент автоматизації.

Обслуговування автоматизованого тестування проводиться для перевірки того, як працюють нові функції, додані в програмне забезпечення: нормально чи ні. Супровід в автотестування виконується, коли додаються нові сценарії автоматизації, і їх необхідно перевіряти і підтримувати, щоб підвищувати ефективність сценаріїв автоматизації з кожним наступним циклом випуску.

Вибір будь-якого інструменту автоматизації може бути заснований на таких критеріях підтримки функціоналу [21]:

– підтримка навколишнього середовища;

– легкість використання;

– тестування бази даних;

– ідентифікація об'єкта;

– тестування зображень;

– тестування відновлення після помилок;

– відображення об'єктів;

– використовуваний мову сценаріїв;

– підтримка різних типів тестування, в тому числі функціонального, тестового управління, мобільного;

– підтримка декількох фреймворків тестування;

– легко налагоджувати сценарії програмного забезпечення автоматизації;

– вміння розпізнавати предмети в будь-якому середовищі;

– великі звіти про випробування і їх результати;

– мінімізація витрат на навчання обраним інструментам.

Застосування засобів автоматизації для тестування програмного забезпечення актуально в наступних випадках:

– Запис в базу даних, логування файлів, backend процеси, тобто місця в системі, що представляють найбільшу важкодоступність.

– Досить часто використовується функціональність, яка має високий рівень ризиків на помилки. При автоматизації критичної функціональності за допомогою тестування гарантовано забезпечується швидке знаходження помилок.

– Автоматизоване тестування ефективно для рутинних операцій. Наприклад, форми, в яких є велика кількість полів для набору даних (перебір даних). Тестовий процес дозволяє автоматично виконувати заповнення полів, а також після збереження здійснювати їх перевірку.

– Даний вид тестування дозволяє автоматизувати процес заповнення полів некоректними даними, проводити перевірку різної валідації.

– Довгі сценарії (end-to-end).

– Тестування даних, для яких необхідні точні математичні розрахунки.

– Для автоматизації правильності пошуку даних.

Залежно від пропонованих замовником вимог і можливостей інструменту, автоматичного тестування підлягають і інші програмні продукти. Розробка окремих тест-кейсів автоматизації робить процес тестування більш ефективним. Створення початкових умов, мінімально впливають на інші тести, дозволяє використовувати сценарії, які здатні повернути систему до початкового стану. При тестуванні з такою функціональністю від автоматизації забезпечується максимальна віддача [22].

Автоматизація тестування є досить складним процесом, що вимагає від розробника тестів певних навичок і вміння. Розробка автоматичних тестів може займати від декількох днів до декількох тижнів, тому роботи по автоматизації тестування повинні бути чітко розплановані і включені в загальний план робіт відділу.

У процесі автоматизації можна виділити три основні етапи:

– початковий етап - етап підготовки і планування;

– етап активної розробки;

– етап підтримки автоматичних тестів.

Схема обміну даними при автоматизації тестування ПЗ наведено на рис.2.3.

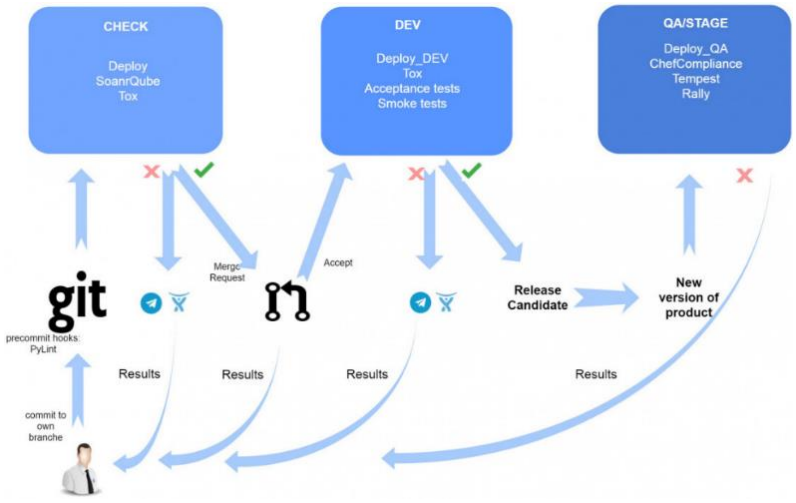

Рисунок 2.3 – Схема обміну даними при автоматизації тестування ПЗ

1. Етап підготовки і планування. На даному етапі відбувається прийняття рішення про необхідність автоматизованого тестування, оцінюються потенційні можливості і економічний ефект, визначаються цілі та стратегії автоматизації, виділяються види тестів, придатні для автоматизації [13].

На цьому ж етапі після ретельного вивчення властивостей, оцінки ступеню відповідності конкретної задачі, а також оцінки ресурсів, необхідних для підтримки нормального функціонування, здійснюється вибір засоби автоматизації.

В результаті початкових розробок створюються вимоги до опису тестів, перевіряється сумісність засобів автоматизації та тестової програмного засобу, а також тестового оточення, виробляються точні методи оцінки витрат на реалізацію. Визначаються стандарти розробки тестових скриптів, посібників, вимоги до апаратного та програмного забезпечення, мережевого оточення, наборам тестових даних.

На даному етапі важливо визначити методи контролю тестового оточення і систему моніторингу дефектів системи.

Заключною стадією першого етапу процесу автоматизації є складання попереднього графіка тестування [14].

2. Етап активної розробки. Переважна частина часу на цьому етапі спрямована на опис, розробку, тестування і запуск автоматичних тестів. Зменшується обсяг ресурсів, необхідних для розробки загальних функцій. У разі великих проектів команда розробників тестів може бути значно збільшена, при цьому успішно завершений етап підготовки і планування гарантує мінімальність ризиків. При розробці тестів важливо передбачити можливість автоматичного документування знайдених дефектів і складання загального звіту про результати виконання автоматичних тестів. Після успішного запуску автоматичних тестів проводиться аналіз отриманих результатів і при необхідності їх доопрацювання. Етап активної розробки може бути досить тривалим залежно від обсягів проекту [17].

3. Етап підтримки автоматичних тестів. Для автоматизації тестування вибираються тільки ті тести, які перевіряють незмінна частина програми. Однак, зміна вимог до вхідних даних, оновлення налаштувань або структури тестового оточення можуть призвести до того, що автоматичні тести будуть видавати помилкові результати. Тому завжди необхідно стежити за змінами в системі і при необхідності коректувати або допрацьовувати автоматичні тести для підтримки їх в актуальному стані.

Для підвищення ефективності автоматизації процесів тестування ПЗ слід [18]:

– Запускати Автотест якомога частіше, щоб швидше реагувати на зміни, швидше знаходити дефекти і стежити за тим, щоб тестове покриття було оптимальним.

– Групувати тести по функціональності в тестові набори, щоб спростити оновлення пов'язаних тест-кейсів при зміні функціональної області.

– Формувати звіти з різними артефактами (скріншоти, логи), щоб можна було легко виявити причину помилки.

– Розробляти тест-кейси таким чином, щоб зміна положення елементів на сторінці не приводило до падіння автотеста і необхідності його поновлення.

– Зберігати тестові дані в файлі конфігурації окремо від кроків Автотест.

– Розробляти тести в наборі таким чином, щоб вони залишалися незалежними один від одного, а невдалий запуск одного з них не впливав на інші.

Піраміда — це дуже зручна метафора, вона наочно показує бажану кількість автоматизованих тестів щодо кожного з рівнів архітектури системи. Повинно бути багато низькорівневих юніт-тестів і зовсім мало високорівневих UI-тестів [21].

Піраміда найпоширеніших типів автоматизації тестування ПЗ наведена на рис.2.4.

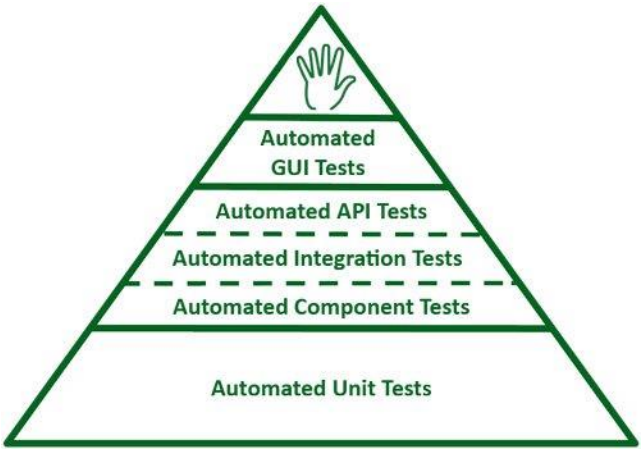

Рисунок 2.4 – Піраміда найпоширеніших типів автоматизації тестування ПЗ

Спочатку розробник вносить нові зміни в код. Тестувальник чекає збірку і деплой нового білда на тестовий стенд. Тестувальник проводить тестування, знаходить проблему і заводить тікет в баг-трекінговій системі. Розробник моментально реагує на цей тікет і виправляє проблему. Це нові зміни в код, і потім знову білд, деплой, ретест.

Важливо розуміти, що юніт-тести тестують код, тобто вони дають розробнику впевненість в тому, що частина його коду працює як задумано, і, що найважливіше, його код не ламає логіку роботи коду його колеги. Це тому, що код колеги теж покритий юніт-тестами, і ці тести розробник запускає перед коммітом у репозиторій.

UI-тести тестують цілісну систему, а саме — те, що буде використовувати користувач. Критично важливо мати такі тести в наявності. Потрібно мати всі види авто-тестів в потрібній кількості на кожному з рівнів. Тоді з’являється можливість отримувати ефективну віддачу від таких тестів.

На рівень API-тестів потрібно опускати всі функціональні тести, які тестувальники проводили протягом спринту. Негативні, позитивні, комбінаторні і т.п. Таким чином створюється швидкий і стабільний пакет регресійних тестів. На рівень UI-тестів виносяться виключно приймальні тести, так звані Happy Path або End-To-End сценарії, які показуються під час демо. Це стосується як веб-, так і мобільних додатків [16].

2.3 Переваги та недоліки автоматизованого тестування

Таким чином, провівши аналітично-дослідну роботу з огляду специфіки та актуальності тематики автоматизації тестування слід зазначити наступні переваги її впровадження в умовах реальних проектів:

– Повторюваність. Розроблені тестові сценарії виконуватися однаково, «людський фактор» не робить на процес несподіваних негативних впливів. Таким чином, тестувальник не зможе пропустити тест.

– Висока швидкість виконання. Не має потреби в звірці етапів тестування з інструкціями і регламентної документації, що істотно зменшує витрати часу на виконання.

– Зниження витрат на підтримку. Підтримка готових тестів і аналіз результатів їх роботи не вимагають таких витрат часу, як проведення цього обсягу роботи вручну.

– Наявність гнучких звітів. Звіти про результати тестування генеруються і розсилаються автоматично.

– Незважаючи на те що для написання тест-кейсів потрібні певні ресурси, окупність у даного підходу для великих і довгострокових проектів може бути величезною. Це можна пояснити тим, що тести легко налаштовуються і можуть багаторазово використовуватися. Впроваджуючи автоматизацію, можна значно знизити вартість кожної години, що витрачається на перевірки, а також знайти найбільш важко виявляються баги.

– Автоматизація може дати можливість швидко зібрати зворотний зв'язок щодо стану програмного продукту і забезпечити більш високу ефективність роботи команди розробників. Це покращує інтеграцію між інженерами по забезпеченню якості, програмістами, менеджерами та іншими учасниками процесу роботи над ПЗ.

– Оскільки автоматизація тестування допомагає збільшити швидкість розробки, вона стає більш рентабельною і значно спрощує виявлення помилки і її усунення на ранній стадії життєвого циклу системи.

– Автоматизація регресійних тестів економить час QA-інженерів і звільняє більше ресурсів для проведення інтуїтивних перевірок та інших активностей по тестуванню. При коректному використанні, автоматизовані тести можуть виконуватися з мінімальною залученістю людини або взагалі без нього.

– Ефективність перевірок багато в чому залежить від якості використовуваних тестових даних. Створювати їх вручну досить ресурсомісткі, тому тестування часто проводиться на копіях «живих» БД. Автоматизовані тести можуть бути спроектовані таким чином, щоб вони створювали і отримували необхідні дані, змінювали їх (не зачіпаючи інші) і поверталися в початковий стан.

– Автоматизація дозволяє проходити етапи проведення тестів швидше, ніж це робить людина. Це особливо актуально для DDT (тестів, керованих даними), оскільки одні й ті ж перевірки проводяться багато разів, але з різними наборами даних.

Недоліками автоматизованого тестування є:

– Повторюваність. Це є і недоліком, що пов'язано з тим, що тестувальник, який виконує тест в ручному режимі, може акцентувати свою увагу на ряді різних деталей, аналіз яких дозволить знайти прихований або не очевидно програмний дефект. Автоматичний скрипт не дозволяє цього.

– Зростання витрат на підтримку. Зі збільшенням кількості релізів і розгортання в різних інфраструктурах підвищуються тимчасові і матеріальні витрати на підтримку.

– Значні витрати на розробку скриптів автоматизації. Це є трудомістким процесом, тому що розробляється додаток, тестуючі іншу програму. У складних і ієрархічних автоматизованих тестах як правило присутні свої прикладні утиліти, бібліотеки і фреймворки, підключення та адаптація яких вимагає значних витрат часу.

– Вартість засоби автоматизації. Ліцензійне ПЗ, його підтримка і установка не є дешевими і оплачуються не один раз. Вільні кошти не мають, як правило, достатньо гнучкості і зручності у використанні.

– Пропуск дрібних помилок. Автоматичний скрипт може не перевірити різні дрібні помилки, перевірка яких не передбачена розробником. До таких помилок відносять різні неточності в позиціонуванні вікон, наявні лінгвістичні помилки в написах, помилки в роботі форм, з якими не проводиться безпосередню взаємодію при виконанні скрипта.

Таким чином, автоматизація процесу тестування є невід'ємною складовою ефективної перевірки та затвердження працездатності сучасного ПЗ за різними критеріями, її впровадження є актуальним в більшості випадків завдяки наявності безлічі переваг. Завдяки використанню існуючих методів, засобів і технологій автоматизації тестування ставити можливим підвищення швидкості і точності виявлення багів в розробленому ПЗ.

3 ДОСЛІДЖЕННЯ ПРОЦЕСУ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ НА ПРИКЛАДІ ВЕБ-САЙТІВ

3.1 Аналіз існуючих аналогів на ринку автоматизації тестування

В даний час на ринку автоматизації процесів тестування веб-сайтів і додатків присутній ряд комерційних і безкоштовних продуктів. Слід зазначити, що серед існуючих засобів автоматизованого тестування найбільш часто на практиці використовуються наступні:

1. SoapUI – це кроссплатформене клієнтське віконне ПЗ з відкритим вихідним кодом, ліцензією GNU і реалізацією на мові програмування (МП) Java. Підтримуються версії для Linux, Windows і MacOS.

Підтримуються технології SOAP / WDSL, REST, Web і HTTP, AMF, JDBC, JMS і системи автоматичного складання Maven, Hudson, Bamboo, Ant, JUnit та інші. Можливості інтеграції з IDE IntellijIdeam, NetBeans, Eclipse.

SoapUI володіє гнучкими можливостями для тестування веб-сервісів шляхом відправки їм повідомлень і отримання відповідей.

SoapUI є одним з провідних функціональних інструментів для тестування SOA і Web-сервісів. За допомогою зрозумілого у використанні графічного інтерфейсу, функцій корпоративного сегмента, це ПЗ забезпечує простоту і мобільність створення і виконання функціональних, регресійних і навантажувальних тестів. Таким чином в рамках єдиного середовища SoapUI забезпечує комплексне покриття тестів - від SOAP і REST, на базі застосування різних веб-служб, до JMS сполучених шарів підприємств, баз даних, Rich Internet Applications [13].

При необхідності, можливо безпосередньо додати WSDL, розробивши зразок запиту для всіх операцій в службі, і створити макет імпортованого WSDL. Після створення проекту надаються функціональні можливості створення і запуску будь-яку кількість функціональних / навантажувальних

тестів і MockServices. Наявність вікна Navigator, що відображає структуру дерева в лівій частині головного вікна, дозволяє візуалізувати прогрес випробування тестів, який постійно знаходиться в полі зору користувача. З головного вікна програми можна управляти і контролювати все, що пов'язано з проектом. Для підвищення рівня автоматизації, SoapUI має у своєму розпорядженні комплекс набору інструментів командного рядка, які забезпечують можливість запуску тестів без використання графічного інтерфейсу.

До функціональних можливостей Soap UI відносяться [7]:

1. Підтримка Mock-сервісів.
2. Використання скриптів з можливістю розширення.
3. Підтримка функціонального, навантажувального тестування.
4. Широкі можливості тестування безпеки ПЗ.
5. Інтеграція з JUnit, Maven, Ant, CI (Hudson & Bamboo).
6. Підтримка плагінів для роботи з IDE: NetBeans, Eclipse, IDEA.

Приклад роботи з тестовим проектом в SoapUI наведено на рис.3.1.

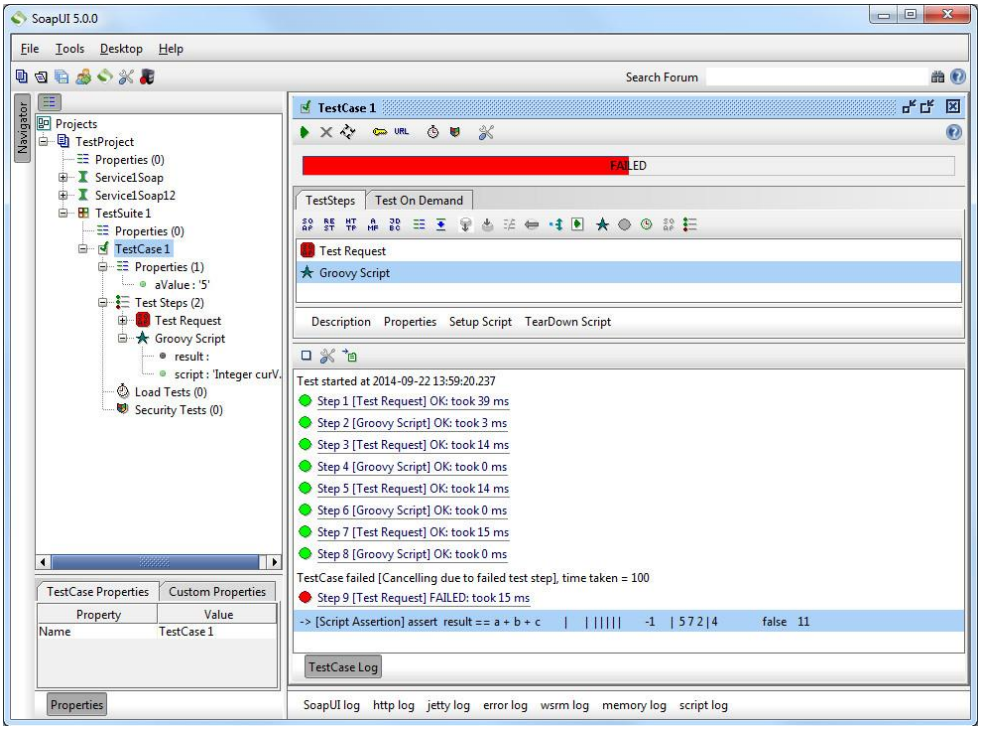

Рисунок 3.1 – Робота з тестовим проектом в SoapUI

Вікно складається з декількох окремих фрагментів:

– Інформація про проекти.

– Налаштування для розглянутого проекту.

– Запит для відправки.

– Отримана відповідь.

Суттєвими перевагами Soap UI є:

– Можливості адаптації WSDL схем для чіткого і зрозумілого сприйняття.

– Підтримка різних заглушок для тестування.

– Гнучкі можливості зберігання і підтримки тестів і тесткейсов [13].

Недоліки:

– Відсутність інтеграції з іншими засобами автоматизації тестування.

– Складна система логування.

– Прив’язка до операційної системи та робочого оточення.

2. Система Selenium включає ряд інструментів автоматизації тестування ПЗ, що мають своє прикладне призначення. Перелічимо основні з них [3].

Selenium Webdriver. Selenium 2 – новий набір інструментів автоматизації тестування, що надає великий набір можливостей щодо забезпечення управління веб-браузером, завдяки наявності цілісного, об'єктно-орієнтованого інтерфейсу, за допомогою чого виключаються різні обмеження, яким часто зустрічалися в ранніх версіях.

Творці WebDriver взяли вирішенні при розробці здійснити злиття двох окремих проектів для створення більш надійного і гнучкого інструменту автоматизації процесу проведення тестування. Як ядро даного засобу може бути інтегрована власна реалізація WebDriver або більш рання, яка була імплементована в рамках Selenium 1. При цьому в Selenium 2 підключена бібліотека Selenium RC, що дозволяє організувати зворотну сумісність. За допомогою даного засобу з'являється підтримка міграції існуючих тестових кейсів на новішу версію інструменту.

Приклад роботи з тестовим проектом в Selenium IDE приведений на рис.3.2.

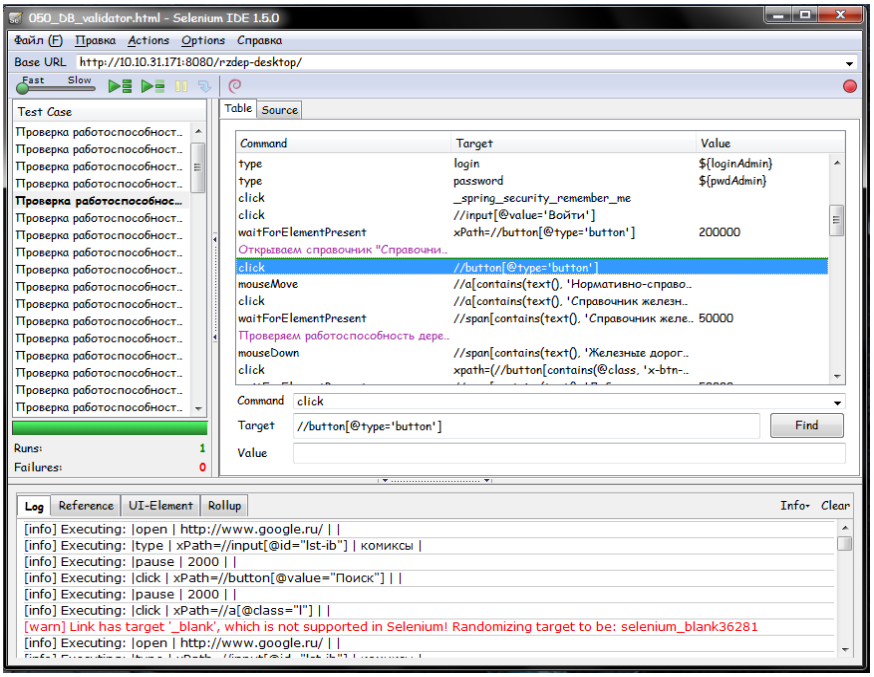

Рисунок 3.2 – Робота з тестовим проектом в Selenium IDE

Selenium Remote Control – перша версія цього інструменту. Selenium 1 широко застосовується на практиці автоматизації тестування ПЗ в режимі супроводу, в силу того, що дозволяє забезпечити ряд можливостей, не до кінця реалізовані в рамках Selenium 2. Зокрема, це підтримка як нових, так і ряду застарілих МП (Perl, Fortran) і підтримку ряду браузерів [8].

Selenium IDE – це середовище розробки, яке застосовується для створення і виконання тестових сценаріїв. Є зручним у використанні доповненням (плагіном) до поширеного безкоштовного браузеру Firefox. Плагін підтримує можливість використання контекстного меню, що забезпечує користувачеві можливість вибору різних елементів інтерфейсу на відкритій сторінці в веб-браузері, після чого є можливість вибору команди зі списку Selenium з необхідними параметрами.

Selenium Grid. Даний інструмент дозволяє забезпечити масштабування значних тестових наборів, підтримуючи при цьому запуск тестів, виконуваних

в декількох наборах оточень. Також він забезпечує можливість запуску тестів в паралельному режимі, що дозволяє різним тестам проходити перевірку одночасно на декількох віддалених робочих станціях або серверах. У зв'язку з цим можна виділити 2 значних переваги цього рішення [35]:

– по-перше, в разі великої кількості тестових операцій або коли тимчасові витрати на імплементацію та перевірку тестів занадто великі, підтримка паралельного режиму сприяє збільшенню продуктивності шляхом поділу тестів на кілька окремих потоків;

– по-друге, в разі, коли розроблені тести ПЗ необхідно запускати в різних робочих середовищах, браузерах або операційних системах, з'являється можливість настройки віддалених серверів розгортання з незалежному запуском одного набору тестів у всіх необхідних середовищах. Це значно прискорює процес тестування та є значною перевагою Selenium в порівнянні з існуючими аналогами [8].

Недоліками даного рішення є необхідність використання повноцінної МП для детальної конфігурації при розробці тестових сценаріїв автоматизації.

3. Ranorex є засобом автоматизації тестування GUI для тестування настільних, веб і мобільних застосувань. Ranorex не має власної мови сценаріїв, і використовує в цій якості стандартні мови програмування C# і VB.NET в якості бази [9, 11].

Підтримувані програми: Windows native (WinForms, WPF, Win32), Java, Qt, Delphi, Flex +, HTML.

Браузери: Internet Explorer, Mozilla Firefox, Chrome, Safari.

Ranorex підтримує можливості фіксації дій на базі застосування інтегрованого рекордера, ідентифікації різних елементів інтерфейсу користувача за допомогою інструменту Ranorex Spy. Всі виявлені елементи зберігаються в форматі XML у відповідних репозиторіях. Окремий елемент в них записаний за допомогою нотації XPath. Виконання тестів відбувається шляхом послідовного запуску .exe файлів test-suite. Після їх виконання формується по одному файлу формату zip на один test-suite, кожен з яких

включає один файл XML з отриманими результатами. Потім програмний скрипт здійснює конвертацію XML формату в xUnit. За рахунок цього досягається можливість отримати звітів по кожному клієнту в Ranorex і в графічному форматі представлення тестів.

На базі записів даних дій відбувається автоматичне формування програмного коду. Кожен крок, при цьому, можна написати вручну [2]. Вікно створення тестового проекту в Ranorex представлено на рис.3.3.

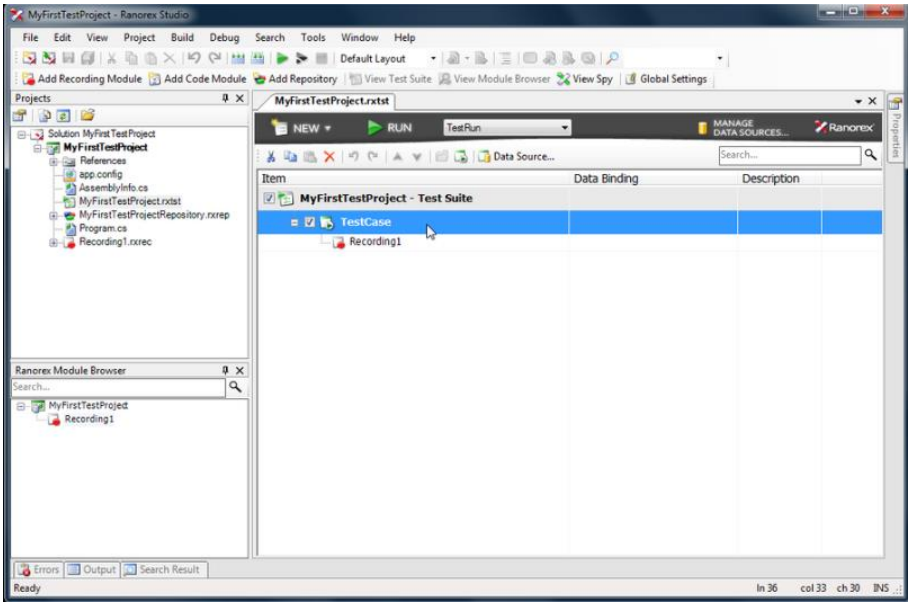

Рисунок 3.3 – Створення тестового проекту в Ranorex

4. IBM Rational Functional Tester - це комплексний інструмент проведення автоматизованого регресійного і функціонального тестування [28].

Це ПЗ надає тестувальникам набір універсальних і зручних механізмів проведення автоматизованого тестування, що застосовуються на практиці для здійснення функціонального, регресійного тестування, а також для тестування користувальницького інтерфейсу.

Вікно роботи з тестовим проектом в Rational Functional Tester представлено на рис. 3.4.

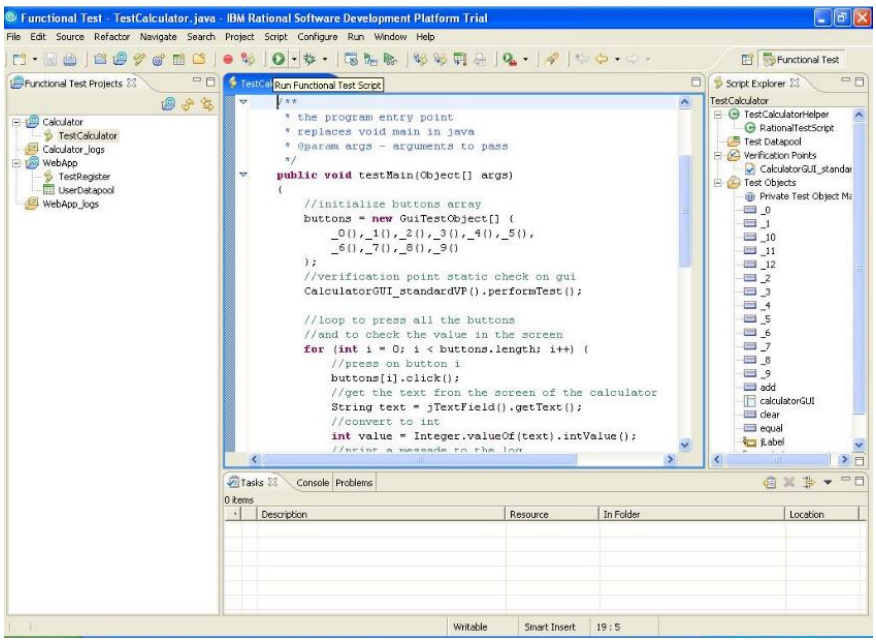

Рисунок 3.4 – Робота з тестовим проектом в IBM Rational Functional Tester

Rational Functional Tester входить до складу IBM Rational Quality Manager, що представляє собою стек засобів управління процесом тестування, ідентифікації дефектів, управління різними версіями сценаріїв здійснення тестування та менеджменту вимог.

Інструментальні засоби, які інтегровані в цю платформу, дозволяють забезпечити процес прискорення розробки ПЗ, що сприяє полегшенню координації та комунікації всередині розробників та тестувальників [30].

Основні переваги [25]:

– проведення процесу тестування ПЗ на мові Java;

– тестування різних веб-застосувань, реалізованих на базі JavaScript, HTML і ін;

– написання модульних тестових сценаріїв на базі мови Java;

– інтеграція технології ScriptAssure для забезпечення перевірки модифікації прикладних даних;

– підтримка комунікаційних колективних засобів системи Rational;

– імплементація можливостей перевірки роботи застосувань 3270 (zSeries) і 5250 (iSeries) за допомогою плагіна Functional Tester Extension для термінальних додатків.

Недоліками є низька популярність у сегменті автоматизації тестування веб-сайтів та складність у засвоєнні функціоналу з різних сценаріїв використання.

5. Empirix - e-TEST suite може використовуватися для комплексного тестування десктопних і веб-застосувань, створених з використанням Microsoft .NET, J2EE, веб-сервісів, а також звичайних застосувань і веб-сайтів. Комплекс включає в себе рішення для різнобічного тестування - від інтерфейсу до тестування навантаження, безпеки і навіть проведення тестів віддалених сайтів в реальних умовах - через файрволи, проксі-сервери [29].

Інший компонент - e-Monitor - в реальному часі збирає статистику з автоматизованого тестування і інформацію про помилки, а також дозволяє планувати тести (наприклад, якщо тестується віддалений сайт, то його можна тестувати в різні періоди часу).

Для тестування інтерфейсу до складу e-TEST suite входить компонент під назвою e-Tester - в оригіналі він називається Automated Regression and Functional Testing Tool for Web Applications (Record / Playback). Цей компонент якраз і призначений для запису певної послідовності дій і відтворення згодом, імітуючи активність користувачів [7]. Таким чином, серед безлічі існуючих засобів немає універсального програмного інструменту, здатного здійснювати підтримку комплексної автоматизації процесу тестування веб-сайтів і додатків. Однак, найбільш доцільним засобом, що дозволяє ефективно інтегруватися з безліччю сучасних мов програмування, є Selenium, в зв'язку з чим його використання буде доцільним в рамках даного дослідження.

3.2 Обґрунтування вибору засобів розробки

Автоматизація тестування являє собою досить складний процес, головною метою якого є створення та подальше вдосконалення (супровід) програмного коду, що володіє високим рівнем надійності і якості. Засоби

програмної автоматизації тестування є деякою сукупністю прийомів, методик, підходів, методів, а також інструментальних програмних додатків (компілятори, бібліотеки, модулі), які використовуються розробником для написання функціонального коду програми, що задовольняє поставленим вимогам. Для дослідження специфіки та вибору засобів реалізації процесу автоматизації, зокрема, мови написання програмного коду, розглянемо ряд популярних і затребуваних мов програмування (МП), що активно використовуються для автоматизації тестування [23].

1. Платформа .NET Framework - це технологія, яка підтримує створення і виконання нового покоління додатків і веб-служб XML. Схема роботи технологій .NET приведена на рис.3.5 [29]. C # - об'єктно-орієнтована МП, розроблена в 1998-2001 роках групою інженерів під керівництвом Андерса Хейлсберг в компанії Microsoft як мова розробки додатків для платформи Microsoft .NET Framework і згодом був стандартизований як ECMA-334 і ISO / IEC 23270.

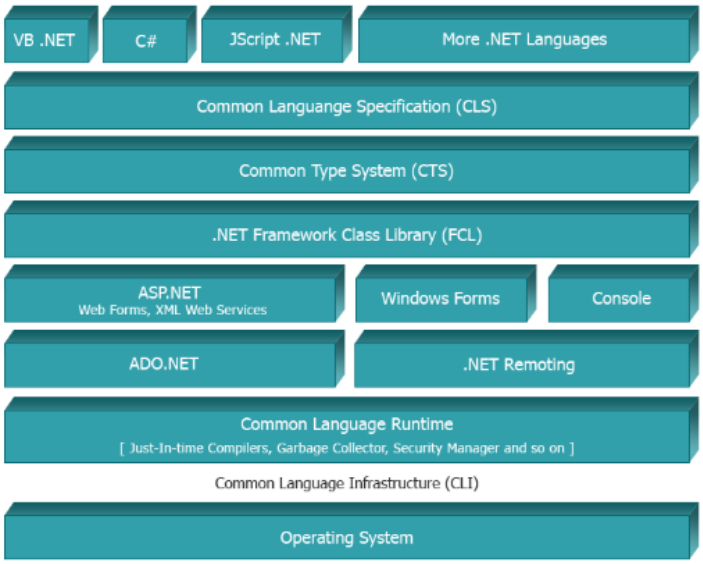

Рисунок 3.5 – Схема роботи технологій .NET

Цілі, які переслідувалися створенням цієї мови, які були сформульовані їм у такий спосіб:

– створити об'єктно-орієнтовану МП, в якій будь-яка сутність є об'єктом, що об'єднує як інформаційну (дані), так і функціональну (дії над цими даними) частини;

– створити першу компонентно-орієнтовану МП сімейства C / C ++;

– спростити C ++, зберігши, по можливості, його функціонал і основні конструкції;

– підвищити надійність коду.

C # відноситься до сім'ї МП з C-подібним синтаксисом, з них її синтаксис найбільш близький до C ++ і Java. Мова має статичну типізацію, підтримує поліморфізм, перевантаження операторів (в тому числі операторів явного і неявного приведення типу), делегати, атрибути, події, властивості, узагальнені типи і методи, ітератори, анонімні функції з підтримкою замикань, LINQ, виключення, коментарі у форматі XML [30].

Перейнявши багато від своїх попередників - мов C ++, Pascal, Smalltalk і, особливо, Java - C #, спираючись на практику їх використання, виключає деякі моделі, що зарекомендували себе як проблематичні при розробці програмних систем, наприклад, C # на відміну від C ++ не підтримує множинне успадкування класів (між тим допускається множинне спадкування інтерфейсів).

Платформа .NET Framework складається з загальномовного середовища виконання (середовища CLR) і бібліотеки класів .NET Framework. Основою платформи .NET Framework є середовище CLR. Середу виконання можна вважати агентом, який керує кодом під час виконання і надає основні служби, такі як управління пам'яттю, управління потоками і віддалена взаємодія. При цьому середовищем накладаються умови суворої типізації та інші види перевірки точності коду, що забезпечують безпеку і надійність.

У Windows Forms термін "форма" - синонім вікна верхнього рівня. Головне вікно програми - форма. Будь-які інші вікна верхнього рівня, які має додаток, також являються формами. Незважаючи на назву, програми для використання Windows Forms, не виглядають як форми. Подібно традиційним

Windows-додаткам програми здійснюють повний контроль над подіями у власних вікнах.

Програми, що використовують Windows Forms, ґрунтуються на класах System.WinForms. Цей розділ включає такі класи, як Form, який моделює поведінку вікон або форм; Menu, який представляє меню; Clipboard, який дає можливість додаткам Windows Forms використовувати буфер обміну. Він також містить численні класи, які надають засоби управління, наприклад: Button, TextBox, ListView, MonthCalendar і т.д. Ці класи можуть бути включені в додаток або з використанням тільки імені класу, або з використанням повного імені, наприклад: System.WinForms.Button. В основі майже кожної програми, написаного із застосуванням Windows Forms, - похідний клас від System.WinForms.Form [31].

Як середовище розробки для мови C # найчастіше застосовується IDE Visual Studio.

Дане рішення реалізується для всіх платформ, які підтримуються вендором засобів розробки Microsoft. До таких платформ відносяться операційні системи Windows 7, 8, 10, Windows Mobile, операційна система в Xbox, а також здійснюється інтеграція Microsoft Silverlight і Azure.

IDE Visual Studio включає в себе гнучкий і сучасний редактор вихідного програмного коду, інтегруючи підтримку технології IntelliSense, а також засоби оперативного профілювання і рефакторінга коду. Наявний в середовищі розробки відладчик здатний функціонувати в своїй якості на рівні вихідного програмного коду, а також є можливості його використання в якості відладчика на машинному рівні. До інших вбудованих засобів та інструментів середовища слід віднести інтуїтивно зрозумілий редактор форм, який сприяє прискоренню процесу створення і конфігурації компонентів графічного інтерфейсу ПЗ, дизайнери класів, об'єктів і схем баз даних.

IDE Visual Studio, також, дозволяє розробляти і інтегрувати в проект сторонні плагіни і функціональні розширення, які забезпечують нові можливості розробки додатків на різних рівнях.

Зокрема, широко використовуються плагіни додавання функцій використання сучасних систем контролю версій (Subversion, Git), інтеграції нових пакетів інструментів для візуального редагування проектування коду на UML-мовою, створення діаграм сценаріїв використання, розробки алгоритмів і ін.

Інтегроване середовище розробки Visual Studio поставляється в різних модифікаціях і може включати наступні основні компоненти: Visual Basic .NET; Visual C ++; Visual C #; Visual F # [32].

2. Java є популярною і затребуваною на ринку праці у сфері автоматизації тестування мовою високого рівня, створеною організацією Sun Microsystems. Офіційну підтримку даної МП здійснює корпорація Oracle. Концепція даного МП заснована на мові C ++, що є базовим при написанні Java.

Створювані на Java додатки збираються в окремий байт-код, виконання якого за допомогою віртуальної машини (JVM) дозволяє інтерпретувати його під конкретну платформу (пристрій) або під будь-яку сучасну операційну систему [33].

Для обходу нестачі Java, що полягає в досить низькій швидкості роботи, застосовуються механізми об'єднання окремих модулів на низькому рівні управління, що дозволяє задіяти мови асемблер, C ++ для вузьких місць, підвищуючи загальний рівень продуктивності.

Java в даний час є одним з найбільш популярних високорівневих МП різних програмних додатків. Підтримкою даної мови сьогодні займається корпорація Oracle. Специфіка синтаксису мови заснована на мові C ++, проте ряд функціональних можливостей відмінний [12].

Програмні додатки компілюються засобами мови в окремий байт-код, що підтримує інтерпретацію під віртуальною машиною Java для заданої апаратно-програмної платформи. Це є певним недоліком, тому реалізований ряд механізмів інтеграції окремих програмних модулів підтримують інші

мови. При створенні мови програмування Java початковими положеннями були [28]:

– простота і гнучкість синтаксису, його зрозумілість і самостійне документування;

– високий рівень продуктивності і захищеність використання виконуваних додатків;

– кроссплатформеність, тобто підтримка роботи на різних програмних платформах і в операційних системах;

– підтримка масштабованості, багатопоточності і роботи з мережевими пристроями.

Схема процесу обробки програмного коду на Java наведена на рис.3.6.

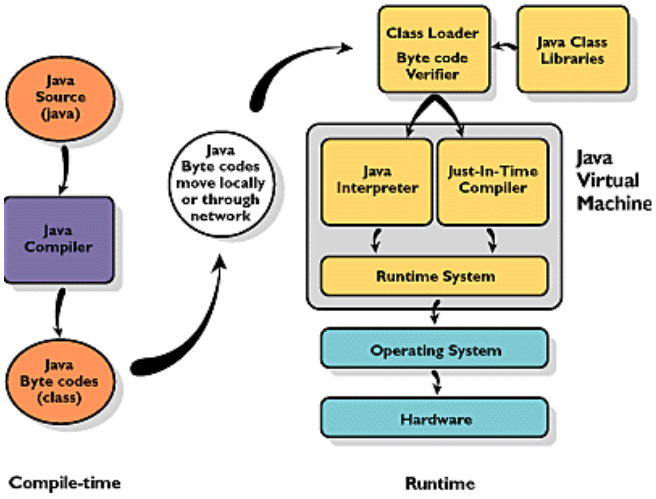

Рисунок 3.6 – Схема процесу обробки програмного коду на Java

В МП всі основні структури даних представляються у вигляді об'єктів. Виняток є лише примітивні типи даних. Всі об'єкти успадковуються від головного об'єкта (Object).

У концепції Java реалізована модель одиничного успадкування щоб уникнути виникнення різних конфліктів між членами класу при реалізації множинного спадкоємства. Це дозволяє спростити їх однозначну ідентифікацію.

В даний час найбільш популярними середовищами розробки для даного ЯПР є Netbeans, Eclipse і Idea [31].

Інтегроване середовище розробки (IDE) NetBeans - безкоштовна система з відкритим вихідним кодом, яка використовується різними розробниками програмних додатків. Дана IDE підтримує всі необхідні засоби, що використовуються для розробки складних десктопних додатків, а також корпоративних систем і веб-сайтів.

NetBeans дозволяє гнучко і оперативно розробляти програми на МП Java, а також підтримує роботу з HTML5, мовою JavaScript і CSS. IDE також надає зручні програмні засоби і модулі для створення програм на ЯП PHP і C / C ++ [33].

Значною перевагою середовища розробки NetBeans є імплементація комплексної підтримки всього стека технологій і засобів МП Java. Історично, це перша вільно поширювана IDE, яка одночасно підтримує JDK 7 і 8, включаючи технології Java EE і фреймворк JavaFX 2.

Інтерфейс середовища написання програмного коду на МП Java Netbeans наведено на рис. 3.7.

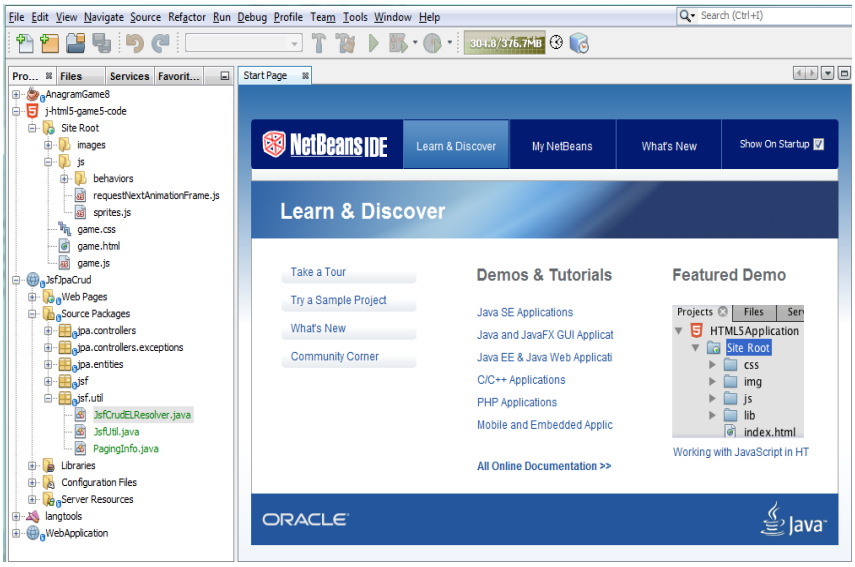

Рисунок 3.7 – Інтерфейс середовища написання програмного коду на МП Java Netbeans

Завдяки постійному розвитку засобів редактора коду, розширенню спектра підтримуваних механізмів розробки та програмних інструментів, підключенню сторонніх шаблонів і модулів, NetBeans є своєрідним стандартом універсальної системи розробки програмного коду.

NetBeans пропонує ряд форм представлення даних, до яких відносяться різні вікна проектів, кошти настройки, конфігурації і управління модулями програми.

Це дозволяє реалізувати оперативний перехід між створеними рівнями ієрархії даних. Також, дана IDE має підтримку систем управління версіями, шляхом інтеграції засобів Subversion, Mercurial і Git «з коробки».

Модуль GUI Builder дозволяє обробляти правильні інтервали між рядками програмного коду, підтримує вирівнювання для Java проектів з урахуванням редагування символів всередині рядка [34].

NetBeans надає розробнику програмні засоби статичного аналізу коду, підтримується робота з широко інструментом FindBugs, для оперативного виявлення і корекції існуючих проблем в коді.

Важливим аспектом роботи відладчика NetBeans є можливість розміщення точок зупину у вікні відображення вихідного коду проекту, додавання контрольних модулів для відповідних полів, а також виконання створеного коду в покроковому режимі з підтримкою створення снепшот для відстеження процесу виконання.

3. Python є об'єктно-орієнтованою високорівневою мовою програмування з підтримкою динамічної типізації, імплементацією процесу автоматичного управління пам'яттю, різних структур даних (кортежі, словники та списки), активно розвивається. Підтримувані об'єкти в Python класифікують на посилальні і атомарні. До останніх відносяться int, long, complex.

При визначенні таких об'єктів здійснюється копіювання їх значень, при цьому в довідкових об'єктах реалізується копіювання тільки покажчика на

об'єкт, в зв'язку з цим, обидві змінні в результаті виконання присвоювання застосовують однакове значення.

Python реалізує створення класів, гнучку і зручну обробку різних виняткових ситуацій, зв'язку модулів і багатопотокові методи обчислень.

Крім ООП мова реалізує структурну, аспектно-орієнтовану, функціональну парадигму програмування [27].

Дана МП дозволяє з'ясовувати тип змінної на етапі виконання програми. Тому, замість присвоювання змінної заданого значення більш коректним є застосування фрази «зв'язування значення з певним ім'ям».

В Python підтримуються вбудовані типи даних: бінарний, Unicode, терміновий, цілочисельний з довільним чином заданим рівнем точності, число з плаваючою комою і інші.

Всі значення є об'єктами, що характерно і для функцій, методів і класів і модулів [31].

Основою мови є CPython, що підтримує значна кількість операційних платформ. Даний інструмент існує під вільною ліцензією, що забезпечує його використання без значних обмежень в додатках.

Розроблено портативні версії даного інтерпретатора для JVM. Створення в програмному коді нового типу даних реалізується шляхом розробки нового класу або формуванням нового типу в спеціальному модулі розширення, який можна реалізувати на одному з підтримуваних мов.

Система класів реалізує повноцінні механізми одиничного і множинного спадкоємства, а також ряд спеціалізованих функцій метапрограмування. Припустимо пряме успадкування від більшої частини вбудованих типів розширень мови.

Кортежі і рядки є колекціями і не можуть бути модифіковані, списки і словники є змінними, які піддаються динамічної модифікації. У зв'язку з цим кортежі є більш продуктивною структурою, ніж списки.

Перевагами даного мови програмування є [32]:

1. Висока интерпретируемость.

2. Стандартизація в динамічному режимі.

3. Гнучка підтримка модульності.

4. Підтримка можливості створення проектів як у функціональному, так і в ООП стилі.

5. Відсутність необхідності в спостереженні за витоками пам'яті, тобто реалізація автоматичної «збирання сміття».

6. Підтримка великої кількості сторонніх і інтегрованих модулів, що розширюють функціонал.

7. Кросплатформеність.

PyCharm - являє собою продуктивну і високоефективну системне середовище розробки для Python з набором різних інструментів і механізмів для організації швидкої розробки програмних додатків.

В даний час є 2 варіанти поставки - PyCharm Community Edition (безкоштовна версія) і PyCharm Professional Edition (платна і розширена версія) [34].

Інтерфейс IDE PyCharm наведено на рис.3.8.

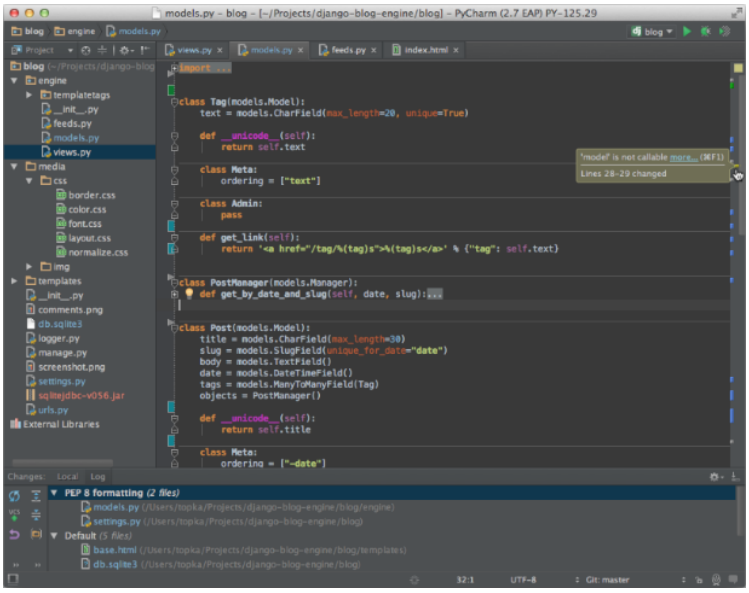

Рисунок 3.8 – Інтерфейс IDE PyCharm для Python

Основні переваги Community Edition:

– Гнучким чином налаштовується для розробки на Python 2.x і 3.x.

– Вихідний код є загальнодоступним і відкритим.

– Підтримка відладчика і функціонального редактора коду.

– Імплементація механізмів багатостороннього рефакторінга коду для інспекції його якості.

– Інтеграція можливостей підключення існуючих систем контролю версій.

– Підтримка засобів оперативної навігації по проекту.

– Підтримка механізмів модульного тестування.

Гнучка настройка користувальницького інтерфейсу для будь-яких потреб розробника.

Після огляду функціональних можливостей наведених засобів розробки проведемо їх варіантний аналіз, для чого застосуємо метод аналізу ієрархій.

Цей метод відноситься до класу критеріальних. Метод полягає в ієрархічній декомпозиції проблеми на все більш прості складові частини і подальшій обробці послідовності суджень експерта по парним порівнянь.

У загальному випадку ієрархічна модель може бути представлена наступним чином: на самому верхньому рівні знаходиться глобальна мета (фокус ієрархії), триває з критеріями, далі до підкритеріям і так далі до самого нижнього рівня - альтернатив.

Після формування ієрархії критеріїв оцінки встановлюються пріоритети (ваги) критеріїв і відповідно до них проводиться оцінка альтернатив за методом лінійної згортки.

В результаті визначається відносна значимість досліджуваних альтернатив для всіх критеріїв, які перебувають в ієрархії.

На підставі суджень експерта будуються матриці парних порівнянь на кожному рівні по відношенню до кожного критерію високого рівня.

Проведемо формалізацію задачі.

Рівень 1 - Вибір засобів розробки.

Рівень 2 - Критерії:

– Гнучкість запуску в операційних системах.

– Затребуваність на ринку автоматизації тестування.

– Обсяг коду для тестування.

– Наявність літератури і кейсів.

Рівень 3 - Альтернативи: Java, C #, Python.

Для фіксації результату порівняння пари альтернатив використовується шкала наступного типу:

– 1 - рівноцінність.

– 2 - помірне перевагу.

– 3 - суттєва перевага.

– 7 - дуже суттєва перевага.

– 9 - висока перевага.

– 2,4,6,8 - проміжні значення.

Результати виконання варіантного аналізу на базі розрахунку парних порівнянь альтернатив для відповідних критеріїв наведено на рис.3.9

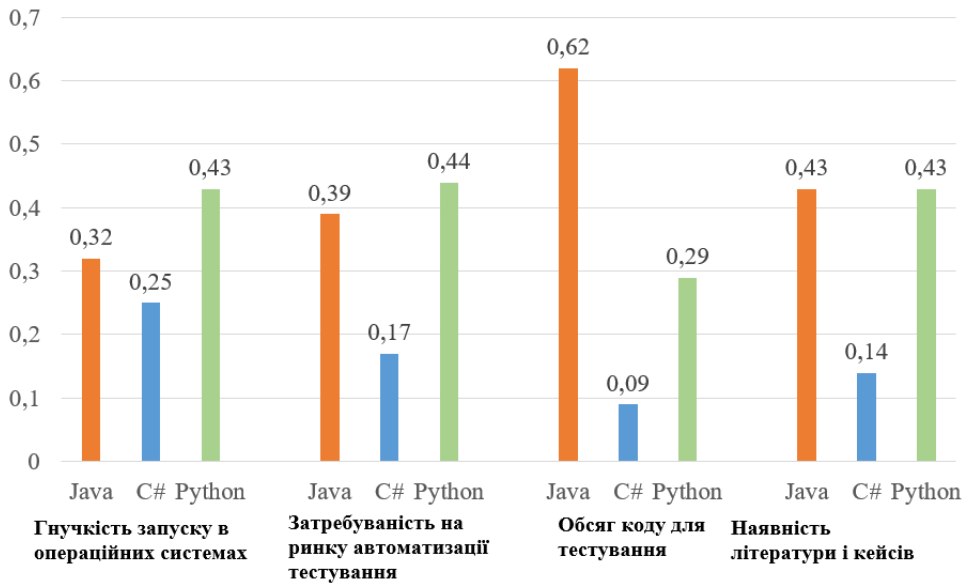

Рисунок 3.9 – Діаграма отриманих результатів порівнянь МП за критеріями

Загальна діаграма отриманих результатів порівнянь МП наведена на рис.3.10.

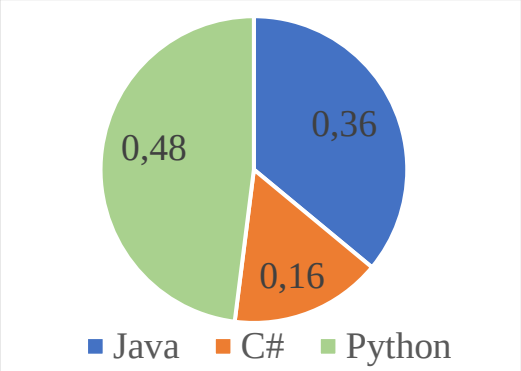

Рисунок 3.10 – Загальна діаграма отриманих результатів порівнянь МП

Таким чином, доцільним є вибір МП Python для автоматизації процесу тестування. Як середовище розробки для даної МП найчастіше на практиці застосовується IDE PyCharm, тому дана система також буде використана.

3.3 Проектування та розробка системи автоматизованого тестування веб-сайтів

Перед початком розробки системи, необхідно виконати її проектування та макетування системи, для визначення функціональних можливостей системи.

Концепція системи складається з необхідності виконувати велику кількість різних програмних тест-кейсів з web-контентом, що сприяє інтенсивному використанні браузера.

Однак для такої роботи необхідно організувати велику кількість різних функціональних рішень, які забезпечать якісну роботу програмної системи. Варіанти використання до інформаційної системи з тестування, зображено на рис 3.11.

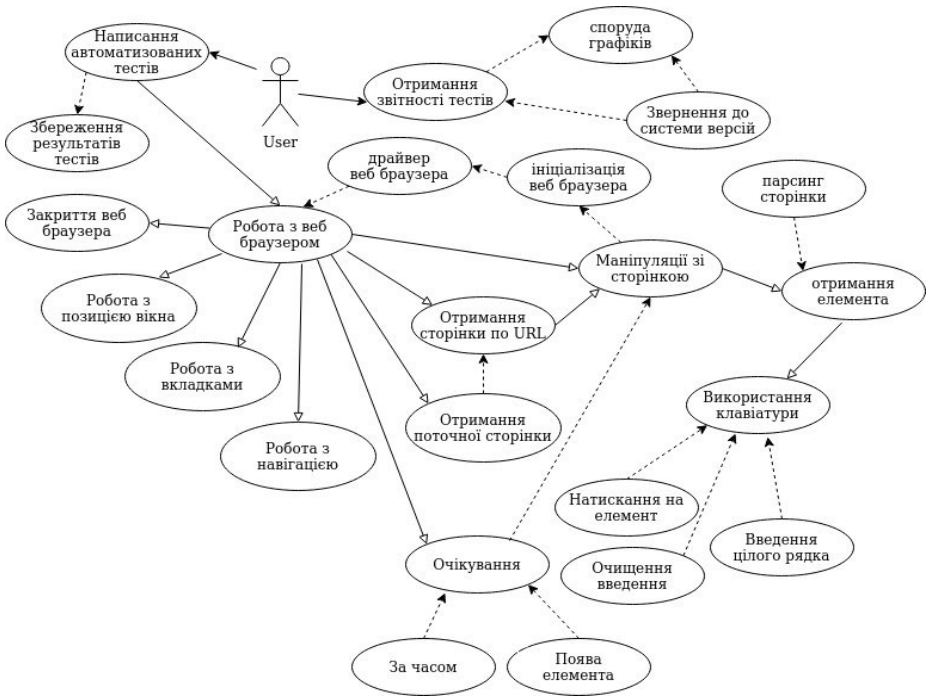

Рисунок 3.11 – Діаграма варіантів використання функціональних можливостей системи

Діаграма варіантів використання охоплює наступні об’єкти:

– Написання автоматизованих тестів – додання до системи редактора та генератора автоматизованих тестових сценаріїв.

– Отримання звітності тестів – спеціалізований модуль електронного документообігу, який дозволяє користувачам системи документувати результати роботи системи.

– Закриття веб-браузера – функціональні відключення роботи браузера, який реалізуються завдяки прямих запитів.

– Споруда графіків – використання спеціалізованих бібліотек для візуалізації результатів тестування.

– Звернення до системи версій – можливість синхронізувати роботу розробленої програмної системи з зовнішніми системами контролю версій для більш надійної роботи програмного засобу.

– Збереження результатів тестів – можливість зберігати отримані тестові результати та тестові сценарії в окремій директорії системи чи в базі даних у вигляді звіту.

– Робота з веб-браузером – функціональне поєднання програмного засобу з роботою браузера для пошуку помилок.

– Драйвер веб-браузера – модуль синхронізації даних для забезпечення роботи з браузером.

– Ініціалізація веб-браузера – отримання зворотного зв’язку стосовно версії браузера та його розширень, які на поточний момент використовуються. А також можливість при запуску наладити систему під необхідність.

– Маніпуляції зі сторінкою – можливість проводити всебічне тестування web-сторінки.

– Отримання елементу – запит на пошук елементу в програмному коді web-сторінки.

– Парсинг сторінки – модуль автоматизованого збору інформації зі сторінки для можливості якісно взаємодіяти з веб-сторінкою та аналізувати вже існуючі компоненти.

– Отримання сторінки за URL – виконання переходу на сторінку з імітацією роботи реального web-сервера.

– Отримання поточної сторінки – швидкий виклик сторінки з виведенням інформації без додаткового завантаження.

– Використання клавіатури – можливість введення даних.

– Натискання на елемент – імітація натискання на об’єкт сторінки та виведення результату її реакції.

– Очищення введення – можливість швидкого очищення усієї введеної інформації на сторінку.

– Введення цілого рядка – можливість додавати одночасно великі обсяги текстової інформації для перевірки розмірності рядків та їх реакції на додану інформацію.

– Очікування – увімкнення системного таймера для перегляду реакції сторінки на роботу системи.

– За часом – увімкнення контролю часу для роботи кожної функції на сторінці системи.

– Поява елемента – швидке виведення елемента на екран.

– Збереження результатів тестів – можливість зберігати робочі сценарії тестування та їх результати роботи.

– Закриття web-браузера – імітація завершення роботи web-браузера.

– Робота з позицією вікна – можливість масштабувати вікно web-браузера для перевірки адаптивності дизайну сторінки.

– Робота з вкладками – можливість імітації роботи декількох вкладок web-браузеру для більш надійної роботи системи.

– Робота з навігацією – можливість переглядати коди елементів та компонентів сторінки.

Після визначення функціональних можливостей системи, необхідно спроектувати логіку роботи кожної з функцій. Для цього використовують діаграми послідовності дії.

Алгоритмічна послідовність дій, являє собою взаємодію - повідомлень між об'єктами, впорядковане з тимчасової осі. На діаграмі послідовності зображені тимчасові послідовності повідомлень.

Діаграма послідовності може існувати у формі дескриптора або ж у формі примірника. Діаграми послідовності і діаграми кооперації зображають, по суті, одну й ту ж інформацію, проте роблять це різним чином. У діаграм такого типу є декілька видів стрілок: синхронні повідомлення (актор передає дію до наступного блока діаграми), відвітна відповідь (такі повідомлення включають в себе якісь дані, або відповідь на запит від попереднього актора), асинхронні повідомлення (актор передав дію до наступного блоку, але все ще виконує сам дії), втрачені та знайдені повідомлення.

Після запуску роботи фреймворка над тест-кейсами за допомогою консолі або готового скрипта починають виконуватися раніше написаний та переданий до виконя окремий тестовий сценарій, або одночас декілька десятків тест-кейсів об’єднаний в один тест-сет.

Фреймворк являє собою драйвер, який дозволяє транслювати написані тестові сценарії з мови програмування Python на Selenium фреймворк, який повністю імітує поведінку браузера. Після завантаження сторінки, драйвер зробить ряд прописаних дій, та зафіксує результат у звіті. Інженер з контролю якості може використовувати автоматизовані чи ручні засоби тестування та оцінки якості як програмного коду сторінки так і функціональних компонентів, або цілих веб-платформ.

Користувач також може у будь-який час переглянути збережені звіти у вибраному каталогу, або за допомогою бази даних, яка має декілька більш інформації чим у звіті.

Графічно кожен об'єкт зображується прямокутником і розташовується у верхній частині своєї лінії життя. Всередині прямокутника записуються ім'я об'єкта та ім'я класу розділені двокрапкою. При цьому вся запис підкреслюється, що є ознакою об'єкта. На рис. 3.12 зображено логіку роботи інженера з системою.

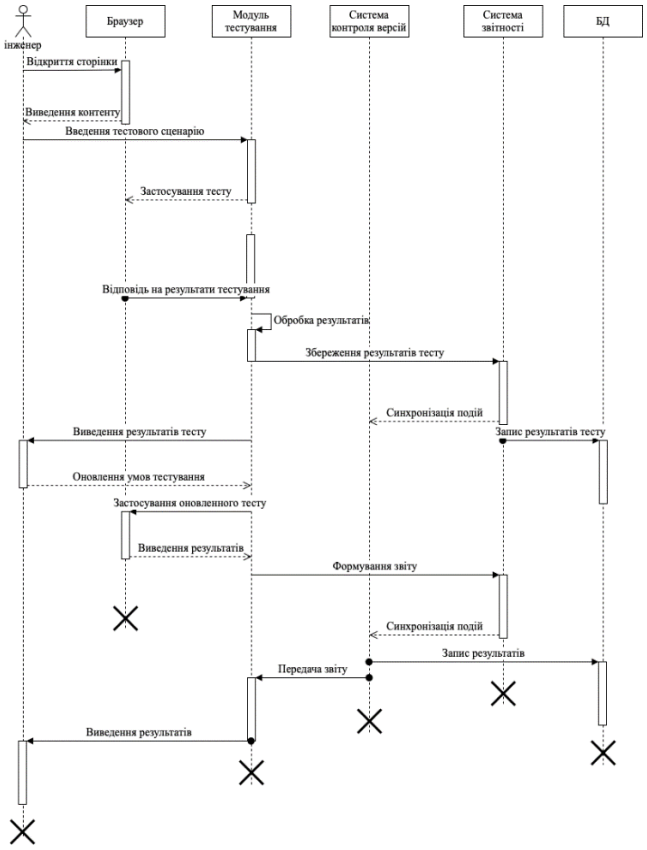

Рисунок 3.12 – Діаграма послідовності дії роботи інженера з контроля якості

Другим виміром діаграми послідовності є вертикальна тимчасова вісь, спрямована зверху вниз. Початкового моменту часу відповідає верхня частина діаграми. Взаємодії об'єктів реалізуються за допомогою повідомлень, які надсилаються одними об'єктами іншим.

В якості системи визначення навантаження на об’єкти програмної системи, використовується взаємодія об’єктів на кооперативній діаграмі. Вона дозволяє продемонструвати роботу кожного об’єкту між собою, а також побудувати активну схему навантаження на кожен об’єкт, рис. 3.13.

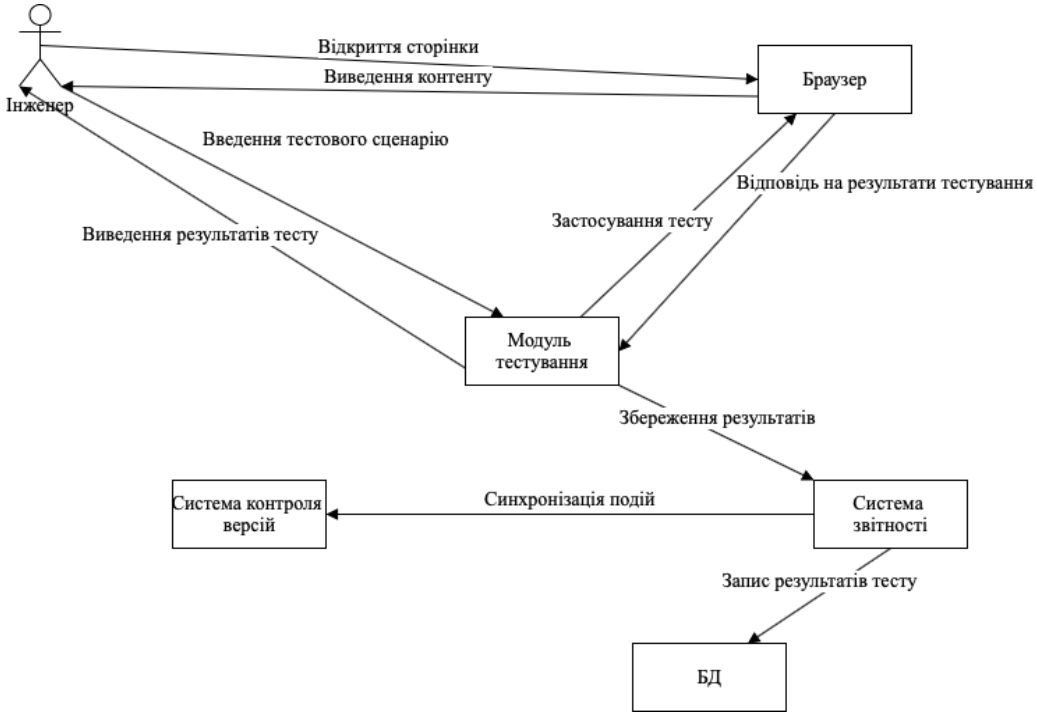

Рисунок 3.13 – Кооперативна діаграма навантаження на об’єкти

На діаграмі зображено взаємодію об’єктів, визначену на діаграмі послідовності діяльності. Самим навантаженим об’єктом програмної системи є браузер та модуль тестування. Кожен з них використовується майже в усіх алгоритмічних діях програмного засобу. Але, оскільки, при взаємодії з цими об’єктами, використовується велика кількість часу (згідно з терміном взаємодії об’єктів в програмній системі), то це є нормою.

Повідомлення зображуються у вигляді горизонтальних стрілок з ім'ям повідомлення, а їх порядок визначається часом виникнення. Масштаб на осі

часу не вказується, оскільки діаграма послідовності моделює лише тимчасову впорядкованість взаємодій типу «раніше-пізніше».

Процес побудування системи класів, для програмного засобу з тестування програмних компонентів, коду та функціональних можливостей, потребує аналізу, використовуємо компонентів системи.

Точка зору реалізації – діаграма класів містить класи, що використовуються безпосередньо в програмному коді (при використанні об'єктно-орієнтованих мов програмування).

В програмному засобі було розроблено наступні класи:

– Клас reportDriver – відповідає за можливість роботи модуля звітування та формування подій, в результаті роботи системи тестування об’єктів.

– Клас conversions відповідає за перетворення рядків у json формат і навпаки. Необхідний для коректної передачі даних в Selenium

– Клас sqliteDriver відповідає за підключення бази даних та синхронізації отриманих результатів тестування з різними функціональними модулями системи, а також з системою контроля за версіями.

– Клас WebGTestClient відповідає за початок роботи. Він запускає сесію яка відповідає за всю програму.

– Клас Chrome відповідає за підключення html сторінок для роботи в імітованому браузері Chrome.

– Клас session – відповідає за роботу з браузером та являє собою інтерфейс, який позволяє запускати та імітувати команди які були описані у тестовому сценарію.

Розроблена система має спеціалізоване призначення, яке дозволяє їй взаємодіяти з web-сторінками, та проводити автоматизоване тестування функціональних можливостей.

Представлення класів, виконано на рис. 3.14.

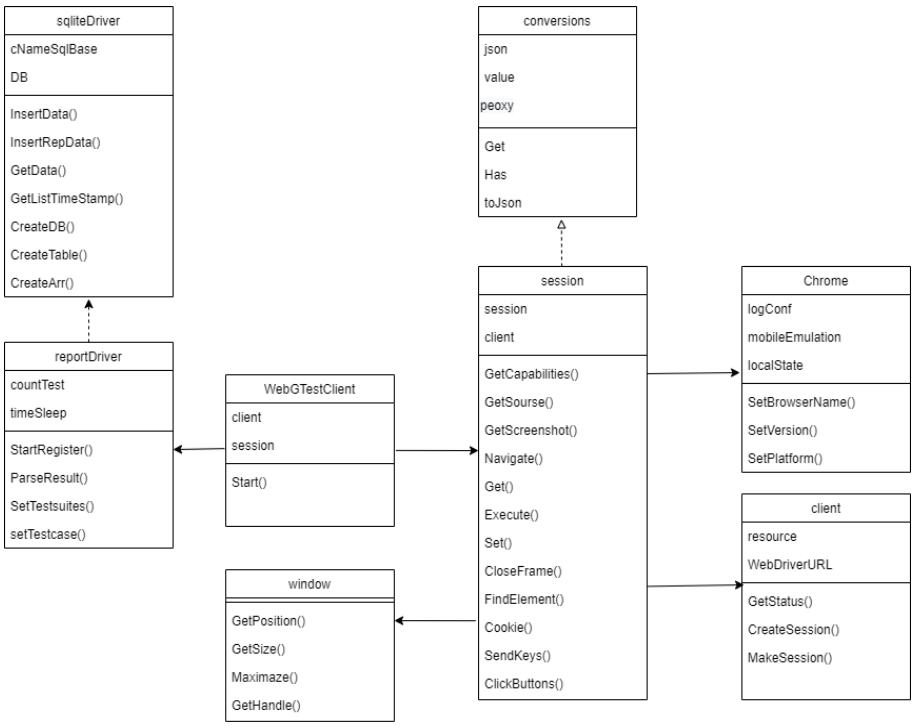

Рисунок 3.14 – Діаграма класів

Таке рішення є актуальною задачею через тестувальників-автоматизаторів, та потребує використання спеціалізованого програмного оточення. До такого оточення слід віднести функціональний сервер Selenium, який дозволяє проводити тестування на хості сайту чи додатку. Однак, Selenium не є досить гнучким рішенням для проведення тестування. Він орієнтований для використання виключно web-мов програмування, що суттєвим чином зменшує ефективність тестування.

Отже, для взаємодії мови програмування Python та серверу тестування Selenium, необхідно розробити спеціалізований драйвер оточення, який дозволить поєднати функціональні можливості та зробити з цього єдине функціональне рішення.

Структурна взаємодія ґрунтується на поєднанні тестових сценаріїв, які написані мовою програмування Python та їх взаємодії з спеціалізованим набором скриптів. Далі тестовий сценарій інтерпретується для виконання на сервері Selenium, який під’єднується до хосту сайту чи додатку. Це дозволяє провести процес тестування, та отримані результати зберегти в оточенні Selenium. Наступним етапом є трансляція результатів тестування в більш

ефективну систему виведення інформації для тестувальника та для інших робітників з групи розробки. Для цього використовується журналізоване формування звіту.

Такий звіт має бути реалізовано в актуальному для перегляду вигляді. Для цього обрана система транслювання результатів в формат HTML і додаватиме до бази даних запис.

З метою більшої візуалізації роботи, виконано розробку спеціалізованої бібліотеки збереження результатів, яка транслює отримані результати до бази даних, після чого використовується бібліотека візуалізації графіки, яка будує діаграми та дозволяє структурувати інформацію в файлі. На рис. 3.15 зображено схему взаємодії системних компонентів в розробленій інформаційній системі.

Розроблене рішення є ефективним інструментом для проведення спеціалізованих автоматичних тестів програмного коду чи візуальних компонентів для web-сайтів чи web-додатків.

При реалізації системи було визначено, що база даних (БД) є необхідною тільки для збереження документообігу, який формується під час роботи системи автоматизованого чи мануального тестування. Це пов’язано з тим, що функціональний модуль написання та використання тестових сценаріїв є самостійним та має використовувати тільки власну інформацію. Внаслідок цього, окрема база даних для збереження тестових сценаріїв не є необхідною.

Однак для збільшення ефективності збереження результатів тестування: в системі використовується база даних SQLite в якій зберігаються події тестування та самі результати у вигляді окремих файлів. Для цього розроблено 3 відповідні таблиці:

– Таблиця data – зберігає в собі дату та час роботи тестового сценарію.

– Таблиця Engeneer – зберігає в собі інформацію стосовно інженера який використовував систему для тестування.

– Таблиця result – зберігає в собі звіти з результатами тестування.

Така БД забезпечую надійну інформацію стосовно роботи програмного засобу, оскілки додатково використовується окремий модуль для тестування, а також система контролю версій для синхронізації діяльності кожного з інженерів.

Системи тестування є досить гібридним видом інформаційних чи програмних систем. Це пов’язано з тим, що для процесу тестування підходять з метою визначити недоліки якогось іншого програмного рішення, та для цього використовують спеціалізовані тестові сценарії.

Взаємодія користувача системи виконується через консольну взаємодію, що пов’язана з можливістю інтеграцією даного фреймворка на CI / CD (комбінація безперервної інтеграції) для автоматичного запуску тестів. На рис. 3.15 зображено консоль з запуском системних тестів.

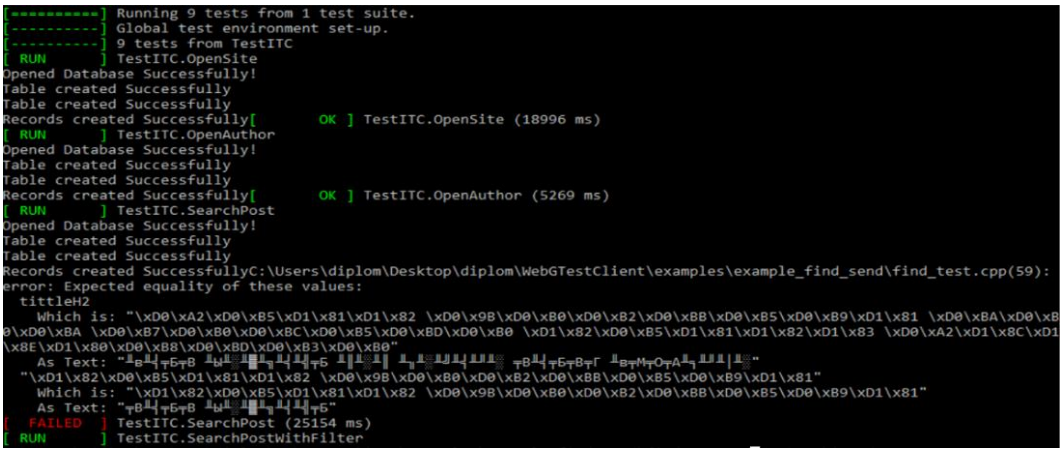

Рисунок 3.15 – Вікно виведення консолі з запуском тестів

Бібліотека передачі тестів використовує скрипт, який виконує пошук активних портів на яких є вихід до веб-серверу.

В більшості випадків, web-середовище використовує однакові порти для функціонування, але якщо потрібно, їх можливо вказати у файлі відповідної конфігурації.

Після визначення порту, виконується передача запиту на сервер Selenium, який дозволяє застосувати набір web-тестів та визначити функціональні особистості роботи web-сайту.

При роботі з тестовим оточенням, використовується система динамічної бібліотеки, яка в реальному часі виконує передачу тестових сценаріїв на віддалений хост, до сесії якого потрібно буде підключитися, або не буде можливості використати та управляти браузером під час тестування. На рис. 3.16 зображено процес роботи Selenium Server та пошуку робочих портів.

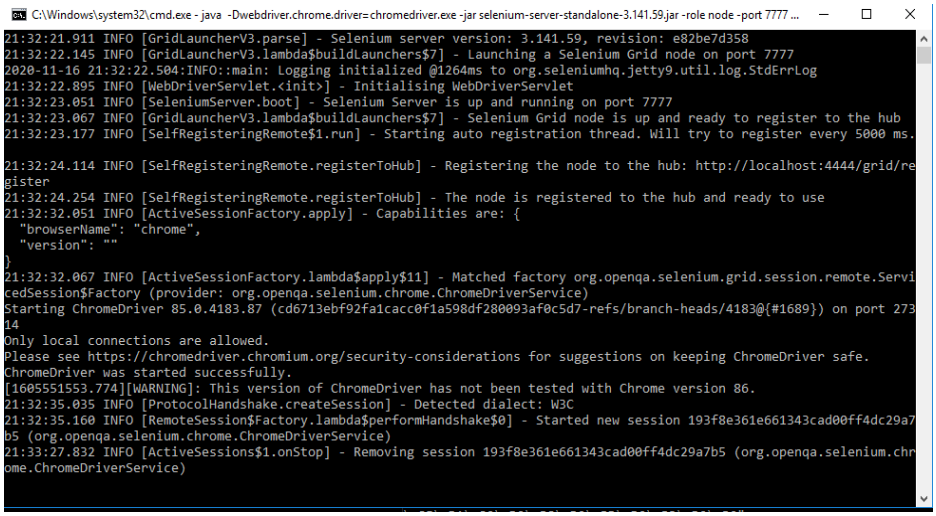

Рисунок 3.16 – Вікно виведення консолі з роботою Selenium Server

Система формування звітів розподілена на декілька функціональних секцій чи зон.

Це дозволяє різним інженерам з оцінки якості продукту, або аналітикам чи іншим, визначати тільки необхідну для них інформацію. Звіт містить окрему зону в який формується інформація про:

– Кількість виконаних тестів.
– Кількість невиконаних тестів
– Можливість обрати період тестування.
– Діаграму ефективності тестів.

На рис. 3.17 зображено виведення візуальної частини звіту за результатами тестування.

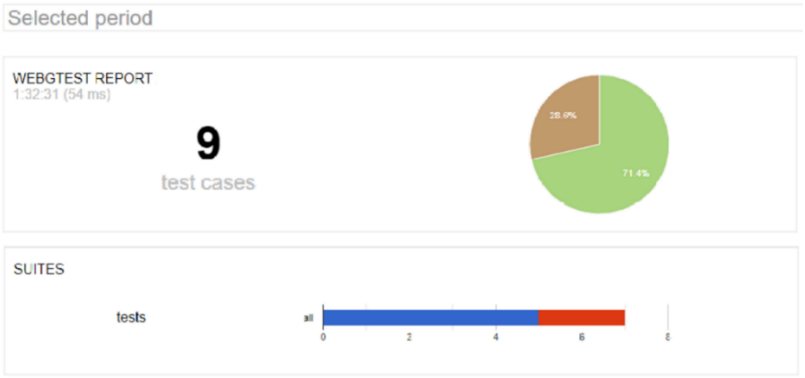

Рисунок 3.17 – Сторінка з виведенням діаграми результатів тестування

Додатково для системи звітування використовується текстовий опис, який дозволяє швидко переглянути результати тестування. У такої форми є:

– Виведення назви тестів.
– Класу тестів.
– Результатів тесту.
– Персонального ідентифікатора тестів.

Для кожного тесту є 3 базові стани його виконання, які використовуються в міжнародному тестуванні та прийняті в якості стандарту:

– Виконаний (complete).
– Невиконаний (failure).
– Пропущений (skipped).

Кожний з цих статусів дозволяє більш деталізовано підходити до тестування будь-якого програмного продукта, та отримувати інформативну інформацію щодо поточного статусу web-системи. Постійне документування та зберігання звітів дозволяє контролювати ефективність роботи з програмними продуктами, а також використовувати архівування звітів на весь період роботи над проектом. На рис. 3.18 зображено сторінку виведення текстової частини звіту.

	Name Test	Class	Result
1	OpenSite	TestITC	completed
2	OpenAuthor	TestITC	completed
3	SearchPost	TestITC	failure
4	SearchPostWithFilter	TestITC	skipped
5	CheckNavigation	TestITC	skipped
6	OnlyForDemo	TestITC	completed
7	OnlyForDemo2	TestITC	completed
8	OnlyForDemo3	TestITC	failure
9	OnlyForDemo4	TestITC	completed

Рисунок 3.18 – Сторінка виведення текстової частини звіту

Бібліотека роботи з тестовими сценаріями використовує окрему директорію в якій необхідно зберігати набори тестових сценаріїв. При роботі з великими тестовими оточеннями, можливо використовувати віддалені чи мережеві репозиторії, з яких інформація з тестовими сценаріями буде передаватися далі для роботи тестового оточення.

Процес тестування включає в себе запуск web-ресурсу, для якого використовуються тестові сценарії.

Бібліотека складається з безліч файлів з класами, корінь містить тільки головні файли, які вже підтягують інші. На рис. 3.19 зображена структура фрагменту проекту у середовищі PyCharm.

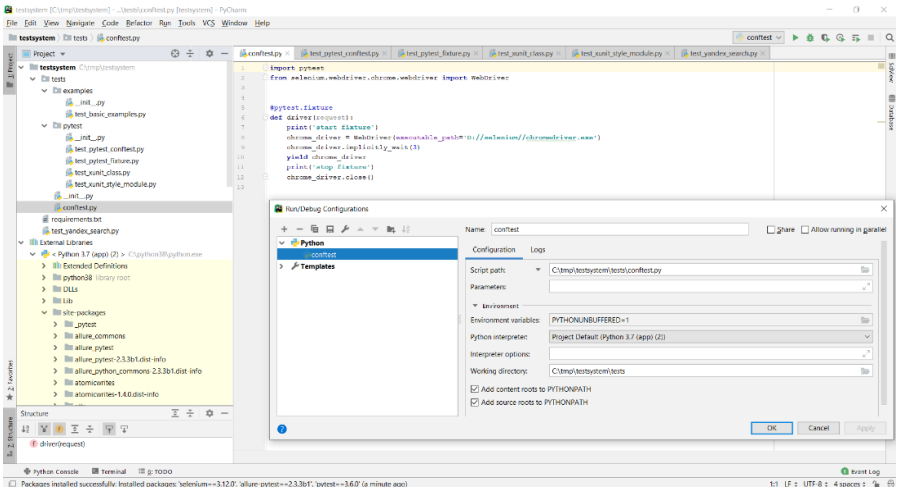

Рисунок 3.19 – Структура фрагменту проекту

Selenium Server відкриває спеціалізований інженерний переглядач web-сторінок, в якому активний трекінг сторінок та верифікація фрагментів коду. Такий переглядач працює окремо від оточення операційної системи, що

дозволяє взаємодіяти з будь-яким контентом без обмежень операційної системи та її політик безпеки. Можна було б використовувати Selenium Grid, який дозволяє навантажувати виконанням тестів лише одну машину з встановленим продуктом, але в даному випадку це буде зайве. На рис. 3.20 зображено вікно з спеціалізованим переглядачем web-контенту.

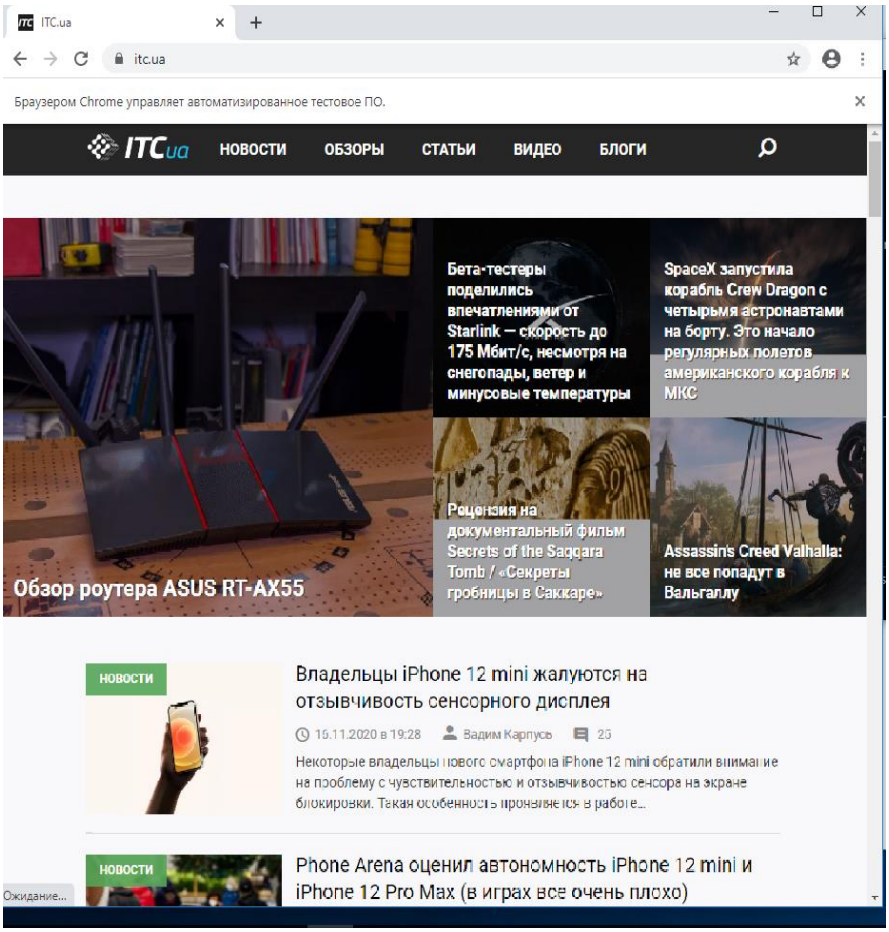

Рисунок 3.20 – Вікно виведення переглядача web-контенту

Однією з головних особистостей роботи системи звітування в розробленому драйвері тестування – є використання засобів оцінки покриття тестовими сценаріями усього програмного коду. Тестове Покриття - це одна з метрик оцінки якості тестування, що вдає із себе щільність покриття тестами вимог або виконуваного коду.

Якщо розглядати тестування як перевірку відповідності між реальним і очікуваним поведінкою програми, що здійснюється на кінцевому наборі тестів, то саме цей кінцевий набір тестів і буде визначати тестове покриття.

Виконання тестового завдання вимагає певного часу. Загальний час тестування визначається кількістю і складністю завдань. Чи повинно це час бути обмеженим або не обмеженої - визначається конкретною ситуацією, в якій застосовується тест.

Довжина тесту (кількість завдань) і час тестування - тісно пов'язані і, в певному сенсі еквівалентні характеристики, але визначальним є все ж саме час тестування, оскільки воно задає поріг втоми, за яким тест починає втрачати свої вимірювальні властивості. Теоретично розрахувати цей час неможливо, тому рекомендується використовувати емпіричні дані за результатами первинної апробації тесту.

На рис. 3.21 зображено виведення результатів тестування з персональними ідентифікаторами тестів, а також час, який було використано для проведення тестування.

Result	Time	Test Id
failure	0.091	1
failure	0.075	2
failure	0.072	3
failure	0.082	4
failure	0.075	5
failure	0.08	6
failure	0.075	7
failure	0.085	8
failure	0.075	9

Рисунок 3.21 – Сторінка виведення звіту з часом витраченим на роботу кожного тесту

Додатково до проекту імпортовано та застосовано модуль звітності Allure Report .Приклад конфігурації звіту через Allure Report у консолі PyCharm наведено на рис.3.22.

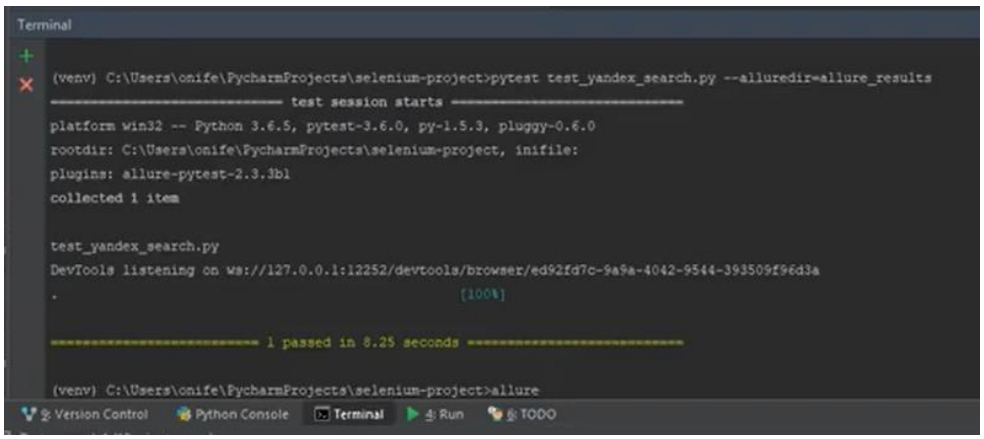

Рисунок 3.22 – Приклад конфігурації звіту через Allure Report

Таким чином, в результаті виконання поставлених завдань у даній роботі здійснено розробку системи підтримки створення та запуску автоматизації тестування веб-сайтів і додатків, виконано дослідження її можливостей і описаний ключовий функціонал. Аналіз отриманих результатів дозволяє стверджувати ефективність застосування сучасних методів, технологій, засобів і мов розробки програмного забезпечення підтримки процесів автоматизації тестування в різних сферах діяльності, що свідчить про значний потенціал і затребуваності на ринку таких рішень.

ВИСНОВКИ

В результаті виконання даної роботи була досягнута її мета, що полягала у закріпленні, розширенні, узагальненні та систематизації знань в рамках досліджуваної предметної дисципліни, за рахунок проведення аналізу специфіки використання сучасних підходів і програмних засобів автоматизації тестування веб-сайтів. Вирішено наступні поставлені завдання даного дослідження:

– Виконано аналіз основних положень та принципів тестування.

– Проведено огляд існуючих типів тестування.

– Здійснено аналіз тестування у життєвому циклу розробки сучасних програмних продуктів.

– Проаналізовано специфіку автоматизованого тестування програмного забезпечення.

– Досліджено процес автоматизації тестування на прикладі веб-сайтів.

В рамках першого розділу наведено результати аналізу особливостей тестування програмного забезпечення, основні положення та принципи тестування програмного забезпечення. Здійснено огляд існуючих типів тестування, надано їх класифікацію та прояснено їх відмінності. Описано місце тестування у життєвому циклу розробки сучасних програмних продуктів.

В рамках другого розділу наведено результати аналізу специфіки автоматизованого тестування програмного забезпечення. Визначено актуальність автоматизації процесу тестування, його базові складові та етапи, наведено основні переваги та недоліки використання та впровадження автоматизованого тестування.

В рамках третього розділу наведено результати дослідження процесу автоматизації тестування на прикладі веб-сайтів. Виконано аналіз існуючих аналогів на ринку автоматизації тестування, проведено обґрунтування вибору

засобів розробки, здійснено проектування та розробку системи автоматизованого тестування веб-сайтів.

Розроблена система може використовуватися для тестування коректності відображення елементів веб-сайтів, їх наявності та розміщення на сторінках, а також для проведення досліджень з отриманих результатів та формування звітності. Подальшим розвитком даного дослідження може бути порівняння існуючих засобів інтелектуальної підтримки процесу автоматизації тестування із застосуванням виявлених переваг в рамках їх впровадження до створеного проекту.

ПЕРЕЛІК ПОСИЛАНЬ

1. Артемов М.А., Караичев С.А. Разработка и стандартизация программных средств и информационных технологий. Разработка и оформление программной документации. Воронеж : ИПЦ ВГУ, 2017. 77 с.

2. Багриновский К.А. Хрусталев Е.Ю. Новые информационные технологии. Москва : ЭКО, 2019. 122 с.

3. Баронов В.В. Информационные технологии. Москва : Компания АйТи, 2014. 512 с.

4. Бахтизин В.В. Стандартизация и сертификация программного обеспечения. Часть 1. Учебное пособие. Минск : БГУИР, 2016. 200 с.

5. Бахтизин В.В. Стандартизация и сертификация программного обеспечения. Часть 2. Учебное пособие. Минск : БГУИР, 2016. 343 с.

6. Бейзер Б. Тестирование черного ящика. Технологии функционального тестирования программного обеспечения и систем. СПб. : Издательский дом "ПИТЕР", 2004. 320 с.

7. Бек К. Экстремальное программирование: разработка через тестирование. СПб. : Питер, 2013. 224 с.

8. Бернс Д. Selenium 2 средства тестирования: руководство для начинающих. Москва : Packt Publishing, 2012. 285 с.

9. Блек Р. Ключевые процессы тестирования. Москва : Академия, 2013. 544 с.

10. Винниченко И.В. Автоматизация процессов тестирования СПб. : Питер, 2005. 203 с.

11. Гагарина Л. Г., Калинников И. С., Фоминова Н. С. Теоретические основы и принципы реализации интеллектуального // *Информационные технологии*. 2018. №8 (144). С. 64-70.

12.Глухова Л.А. Технологии разработки программного обеспечения Учебное пособие. Минск : Белорусский государственный университет информатики и радиоэлектроники, 2013. 178 с

13.Гостомыслов К., Гостомыслов Л. Тестирование: плюсы и минусы. // *Высшее образование*. 2011. №3. С. 152-154.

14.Дастин Э., Рэшка Д., Пол Д., Автоматизированное тестирование программного обеспечения. Внедрение управление и эксплуатация. Москва : ЛОРИ, 2003. 567 с.

15.Калбертсон Р. Быстрое тестирование. Москва : Издательский дом «Вильяме», 2002. 384 с.

16.Канер С. Тестирование программного обеспечения. Киев : ДиаСофт, 2014. 612 с.

17.Ковалев С.Н. Информационные технологии Учебное пособие. Москва : Московский гос. ин-т радиотехники, электроники и автоматики (технический университет), 2015. 147 с.

18.Ковалевская Е.В. Метрология, качество и сертификация программного обеспечения. Москва : МЭСИ, 2017. 38 с.

19.Котов С.Л., Палюх Б.В. Разработка, стандартизация и сертификация программных средств и информационных технологий и систем. Учебное пособие. Тверь : ТГТУ, 2016. 104 с.

20.Липаев В.В. Тестирование программ. Москва : Радио и связь, 1986. 296 с.

21.Липаев В.В. Сертификация программных средств. Москва : СИНТЕГ, 2019. 348 с.

22.Машкин М.Н. Информационные технологии. Москва : ВГНА, 2013. 200 с.

23.Минитаева А.М. Разработка и стандартизация программных средств и информационных технологий. Омск : Изд-во ОмГТУ, 2018. 92 с.

24.Ноздрин В.С. Разработка и стандартизация программных средств: учебное пособие. Москва : МГИУ, 2019. 46 с.

25.Романова Ю.Д. Информатика и информационные технологии. Москва : Эксмо, 2008. 592 с.

26.Себеста Р. Основные концепции языков программирования. Москва: Эксмо, 2013. 316 с.

27.Стотлемайер Д. Тестирование web - приложений. Москва : КУДИЦ-ОБРАЗ, 2013. 315 с.

28.Тамре Л. Введение в тестирование программного обеспечения. Москва : Издательский дом «Вильяме», 2003. 368с.

29.Тюгашев А.А.. Основы программирования. Часть 1. СПб.: Университет ИТМО, 2016. 160 с.

30.Тюгашев А.А.. Основы программирования. Часть 2. СПб.: Университет ИТМО, 2016. 160 с.

31.Фильчаков В.В. Стандартизация жизненного цикла и качества программных средств. СПб. : СПбГУАП, 2012. 330 с.

32.Холодов Г.М., Поповкин А.В. Алгоритмическое и объектно-ориентированное программирование. Москва : МГТУ МАМИ, 2013. 459 с.

33.Хорстманн С. Java. Библиотека профессионала. Москва : ООО И.Д. Вильямс, 2014. 864 с.

34.Хлебников А.А. Информационные технологии. Москва : КноРус, 2016. 466 с.

35.Коваленко Д. Selenium Design Patterns and Best Practices. Москва : PACKT Publishing, 2014. 320 с.

ДОДАТОК А

Приклад фрагменту реалізації системи

```
import allure
from allure_commons.types import Severity
from selenium.webdriver.chrome.webdriver import WebDriver
from selenium.webdriver.support.wait import WebDriverWait

@allure.title('Search returned more than 10')
@allure.severity(Severity.BLOCKER)
def test_yandex_search():
    driver = WebDriver(executable_path='D://selenium//chromedriver.exe')
    with allure.step('Open search page'):
        driver.get('https://ya.ru')

    with allure.step(Шукаємо market'):
        search_input = driver.find_element_by_xpath('//input[@id="text"]')
        search_button =
driver.find_element_by_xpath('//div[@class="search2__button"]//button[@type="submit"]')
        search_input.send_keys('market.yandex.ua')
        search_button.click()

    def check_results_count(driver):
        inner_search_results = driver.find_elements_by_xpath('//li[@class="serp-item"]')
        return len(inner_search_results) >= 10

    with allure.step(Expect more test results 10'):
        WebDriverWait(driver, 5, 0.5).until(check_results_count, 'The number of search results is less 10')

    with allure.step('Follow the link of the first result'):
        search_results = driver.find_elements_by_xpath('//li[@class="serp-item"]')
        link = search_results[0].find_element_by_xpath('.//h2/a')
        link.click()

    driver.switch_to.window(driver.window_handles[1])
    with allure.step('Check the correctness of the Title of the page '):
        assert driver.title == 'Yandex.Market - selection and purchase of goods from trusted online stores'
```

```
import allure
from selenium.webdriver.chrome.webdriver import WebDriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions
from selenium.webdriver.support.wait import WebDriverWait


def test_check_data_on_page():
    1.open lamoda
    2.select the region
    3.open the delivery details page
    4.Change delivery city
    5.check that there are 6 types of delivery, check their prices, and delivery times    """
   initial_city = 'Киев'
   change_city = 'Львов'

   expected_shipment_data = [
      {
         'name': 'Без примерки',
         'price': 'Бесплатно',
         'shipment_term': 'На следующий день',
         'fitting': 'Отсутствует',
         'payment': 'Наличными или банковской картой'
      },
      {
         'name': 'С примеркой',
         'price': 'Бесплатно',
         'shipment_term': 'На следующий день',
         'fitting': 'Время на примерку одного заказа - 15 минут.\nВозможна при заказе до 10 товаров.',
         'payment': 'Наличными или банковской картой'
      },
      {
         'name': 'Сегодня, с примеркой',
         'price': '300 грн.',
         'shipment_term': 'В день заказа',
         'fitting': 'Время на примерку одного заказа - 15 минут.\nВозможна при заказе до 10 товаров.',
         'payment': 'Наличными или банковской картой'
      },
      {
         'name': 'В течение выбранного часа, с примеркой',
         'price': '350 грн.',
         'shipment_term': 'На следующий день',
```

```
        'fitting': 'Время на примерку одного заказа - 15 минут.\nВозможна при заказе до 10 товаров.',
        'payment': 'Наличными или банковской картой'
    },
    {
        'name': 'В течение выбранных 15 минут, с примеркой',
        'price': '800 грн.',
        'shipment_term': 'На следующий день',
        'fitting': 'Время на примерку одного заказа - 15 минут.\nВозможна при заказе до 10 товаров.',
        'payment': 'Наличными или банковской картой'
    },
    {
        'name': 'В течение выбранного часа, с примеркой',
        'price': грн руб.',
        'shipment_term': 'На следующий день',
        'fitting': 'Время на примерку одного заказа - 15 минут.\nВозможна при заказе до 10 товаров.',
        'payment': 'Наличными или банковской картой'
    }
]

driver = WebDriver(executable_spath='D://selenium//chromedriver.exe')
driver.implicitly_wait(3)
with allure.step('Open https://www.lamoda.ru/'):
    driver.get('https://www.lamoda.ru/')
```