УДК 004.9
DOI https://doi.org/10.32689/maup.it.2021.1.10

Денис ШИБАЄВ
аспірант кафедри технічної кібернетики та інформаційних технологій, Одеський національний морський університет, вул. Мечникова, 34, м. Одеса, Україна, індекс 65029 (denscreamer@gmail.com)

ORCID: https://orcid.org/0000-0002-3260-5843

Наталя ШИБАЄВА
кандидат технічних наук, доцент кафедри інформаційних технологій, Державний університет «Одеська політехніка», просп. Шевченко, 1, м. Одеса, Україна, індекс 65001 (nati.shibaeva@gmail.com)

ORCID: https://orcid.org/0000-0002-7869-9953

Микола РУДНІЧЕНКО
кандидат технічних наук, доцент кафедри інформаційних технологій, Державний університет «Одеська політехніка», просп. Шевченко, 1, м. Одеса, Україна, індекс 65001 (nickolay.rud@gmail.com)

ORCID: https://orcid.org/0000-0002-7343-8076

Володимир НІКІФОРОВ
студент, ПрАТ «ВНЗ «Міжрегіональна Академія управління персоналом», вул. Фрометівська, 2, м. Київ, Україна, індекс 03039 (nikifvova@gmail.com)

ORCID: https://orcid.org/0000-0001-6241-1516

Denis SHIBAYEV
Graduate student of the Department of Technical Cybernetics and Information Technologies, Odessa National Maritime University, str. Mechnikova, 34, Odessa, Ukraine, postal code 65029 (denscreamer@gmail.com)

Natalia SHIBAYEVA
Candidate of Technical Sciences, Associate Professor of Information Technology, Odessa Polytechnic State University, ave. Shevchenko, 1, Odessa, Ukraine, postal code 65001 (nati.shibaeva@gmail.com)

Mykola RUDNICHENKO
PhD in Technical Sciences, Associate Professor of Information Technology, Odessa Polytechnic State University, ave. Shevchenko, 1, Odessa, Ukraine, postal code 65001 (nickolay.rud@gmail.com)

Volodymyr NIKIFOROV
Student, Interregional Academy of Personnel Management, str. Frometivska, 2, Kyiv, Ukraine, postal code 03039 (nikifvova@gmail.com)

Бібліографічний опис статті: Шибаєв Д., Шибаєва Н., Рудніченко М., Нікіфоров В. Проектування гнучкої системи тестування web-ресурсів. *Інформаційні технології та суспільство*. 2021. Вип. 1. С. 85–91. DOI: https://doi.org/10.32689/maup.it.2021.1.10

Bibliographic description of the article: Shybaiev, D., Shybaieva, N., Rudnichenko, M., Nikiforov, V. (2021). Proektuvannia hnuchkoi systemy testuvannia web-resursiv [Design of a flexible web-resource testing system]. *Informatsiini tekhnolohii ta suspilstvo – Information technology and society*, 1, 85–91. DOI: https://doi.org/10.32689/maup.it.2021.1.10

ПРОЕКТУВАННЯ ГНУЧКОЇ СИСТЕМИ ТЕСТУВАННЯ WEB-РЕСУРСІВ

Анотація. *Роль тестування є невід'ємною частиною в сучасних процесах розробки, більш того, ця роль суттєво збільшується. Це пов'язано зі зростанням складності програмного продукту і збільшення вимог до його якості. З'являються нові способи і методи розробки програмного забезпечення та його підтримки. Одним з таких підходів є впровадження різних видів тестування під час етапу розробки продукту для виявлення і мінімізації програмних і логічних помилок, та підвищення якості програмного коду і продукту взагалі. Існує безліч методологій і типів тестування відмінності яких знаходиться в цілях і об'єктах тестування. Регресійне тестування, яке направлено на виявлення*

дефектів у вже протестованих ділянках коду, в основному проводять автоматизованим тестуванням. Для цього використовують системи з тестування програмного коду і пошуку помилок, які є програмними драйверами. Вони дозволяють аналізувати інформацію яка використовується в системі та методом перегляду програмного коду виконувати аналіз його працездатності. ***Метою*** *статті є розробка проекту системи тестування web-ресурсів в реальному часі із застосуванням гнучкого редактора тестових сценаріїв та системи масштабування функціональних можливостей системи. Реалізація поставленої мети передбачає вирішення низки* ***завдань****: 1) проектування системи обробки тестових сценаріїв; 2) формування системної логіки та поведінки програмної системи; 3) проектування аналітичного модуля візуалізації результатів тестування.* ***Наукова новизна.*** *Спроектоване рішення є спеціалізованим драйвером до системи тестування та перевірки якості Selenium. Таке рішення дозволяє тестувати web-ресурси із спеціалізовано-спроектованого тестового простору та аналізувати отримані результати із застосуванням спеціалізованого аналітичного модуля із візуалізацією графічних компонентів. Як* ***висновок****, у статті наголошується, що розробка сучасного програмного засобу, який дозволить верифікувати, аналізувати та перевіряти програмний код на наявність помилок для використання з прикладним програмуванням є актуальною та сучасною задачею, яка суттєвим чином підвищить якість розробки програмних продуктів та забезпечить можливість зберігати робочу документацію в єдиному вигляді. Подальшим розвитком є інтеграція окремого редактора тестових сценаріїв та додання можливостей проведення різних типів тестування програмних систем.*

Ключові слова: *тестування, selenium, якість програмних рішень, аналіз якості, web-системи.*

DESIGN OF A FLEXIBLE WEB-RESOURCE TESTING SYSTEM

Abstract. *The role of testing is an integral part of modern development processes, moreover, this role is significantly increasing. This is due to the increasing complexity of the software product and increasing requirements for its quality. New ways and methods of software development and support appear. One such approach is the introduction of various types of testing during the product development phase to detect and minimize software and logical errors, and improve the quality of software code and the product in general. There are many methodologies and types of testing, the differences of which are in the goals and objects of testing. Regression testing, which aims to detect defects in already tested areas of the code, is mainly performed by automated testing. To do this, use systems for testing software code and debugging, which are software drivers. They allow you to analyze the information used in the system and the method of viewing the program code to analyze its performance.* ***The aim.*** *The aim of the article is to develop a project of a system for testing web-resources in real time using a flexible test script editor and a system for scaling the functionality of the system. Realization of the set purpose provides the decision of a number of tasks: 1) designing of system of processing of test scenarios; 2) the formation of system logic and behavior of the software system; 3) design of an analytical module for visualization of test results.* ***Scientific novelty.*** *The designed solution is a specialized driver for the Selenium testing and quality control system. This solution allows you to test web-resources from a specialized-designed test space and analyze the results using a specialized analytical module with visualization of graphical components. In* ***conclusion,*** *the article emphasizes that the development of modern software that will verify, analyze and verify the program code for errors for use with application programming is an urgent and modern task that will significantly improve the quality of software development and ensure the ability to store working documentation in a single form. Further development is the integration of a separate test script editor and the ability to conduct various types of software testing.*

Key words: *testing, selenium, quality of software solutions, quality analysis, web-systems.*

Актуальність проблеми. Інформаційні системи з точки зору системного аналізу є складними системами, оскільки складаються з безлічі різних частин: функціональні модулі, сервера різного напрямку, бази даних, методів передачі даних і іншого. В спроектованих системах з безліччю взаємозв'язків між різними компонентами, інформаційних систем (ІС) один з одним 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 відмовостійкість 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.

В життєвому циклі програмного продукту передбачено тестування програмного забезпечення (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 життєвий цикл 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 фактора.

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

Комплексне тестування обернено на пошук невідповідність 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 чорній коробки [3–5].

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 [6].

Аналіз останніх досліджень і публікацій. Завдяки веб-платформам не тільки створюється перше враження про компанію або продукт у відвідувачів, але ще може відбуватися безліч важливих дій, таких як покупки продуктів або управління об'єктами в режимі реального часу. Саме тому в веб платформах величезне значення має надійність, функціональність і зручність сайту.

Проаналізувавши предметну область використання програмних засобів для проведення тестування програмного коду і робот інформаційних систем, було визначено, що веб платформи з тривалим життєвим циклом потребують тестування функціоналу та коду, щоб домогтися надійності системи і правильного функціонування всіх модулів.

Також можна прийти до висновку, що автоматизація тестування істотно зберігає витрати на завданнях такого плану. Адже з'являється можливість заощадити ресурси на відсутності дублювання ручного тестування при постійно зростаючому функціонал і релізів [7–8].

Можна суттєво зменшити витрати на додаткових фахівців якщо забезпечити можливість написання коду на зручному для тестувальника мовою програмування. Для цього необхідно забезпечити і синхронізувати роботу з системою на основі розробленого драйвера до чинного способу управління веб-браузера через методи запитів. Така система дозволить не витрачати безліч фінансових ресурсів на розробку всієї бібліотеки автоматизованого тестування куди входить управління браузером. Так само, залишається можливість розробити систему модульної, що істотно чином дозволить застосувати будь-яке програмне рішення до програмного коду. Реалізацію програмного засобу слід організувати з використанням сучасних технологічних рішень та алгоритмів [9–15].

Таким чином, при реалізованої системи автоматизованого тестування веб-додатків можна заощадити фінанси, якщо під час життєвого циклу інформаційної системи необхідно виконувати регресивні, димчаті тести раз по раз при кожному релізі. Або в ІС буде безліч модулів, які взаємодіють між собою, в такому випадку без автоматизованих тестів обійтися не вийде. Підкріплює це все можливість складання автоматичної звітності.

Метою статті є проектування інтерактивного драйверу для системи sileneum, яка дозволить проводити автоматизоване тестування web-ресурсів із збереженням результуючих звітів та можливістю аналізу протестованих фрагментів.

Виклад основного матеріалу. Концепція системи складається з необхідності виконувати велику кількість різних програмних тест-кейсів з web-контентом, що сприяє інтенсивному використанні браузера. Розробка має сугубо спеціалізований напрям, отже таке програмне рішення впроваджується тільки до спеціалізованих проектів та використовують його спеціалісти з забезпечення якості програмного коду та роботи системи. Таким чином, використання відбувається тільки персоналізовано без авторизації або реєстрації в системі.

Першим етапом є створення діаграми варіантів використання, завдяки якій, визначаються усі вимоги до кінцевої системи. Після виконання всіх етапів проектування, здійснюється перехід до стадії розробки програмного забезпечення.

Діаграма варіантів використання, рис. 1, охоплює частину наступних об'єктів:

– 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 організовувати введення даних.

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

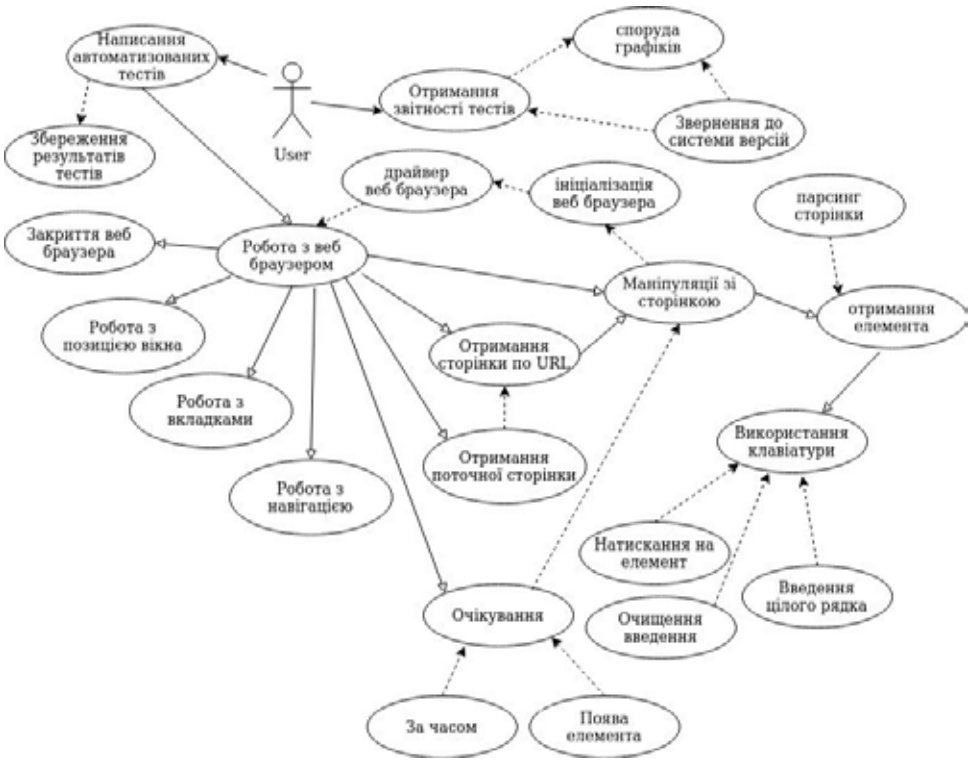

Рис. 1. Діаграма варіантів використання

– 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.

Після визначення функціональної складової системи, виконується проектування логічних компонентів системи, а саме класів, які складають загальну концептуальну логіку системи, рис. 2.

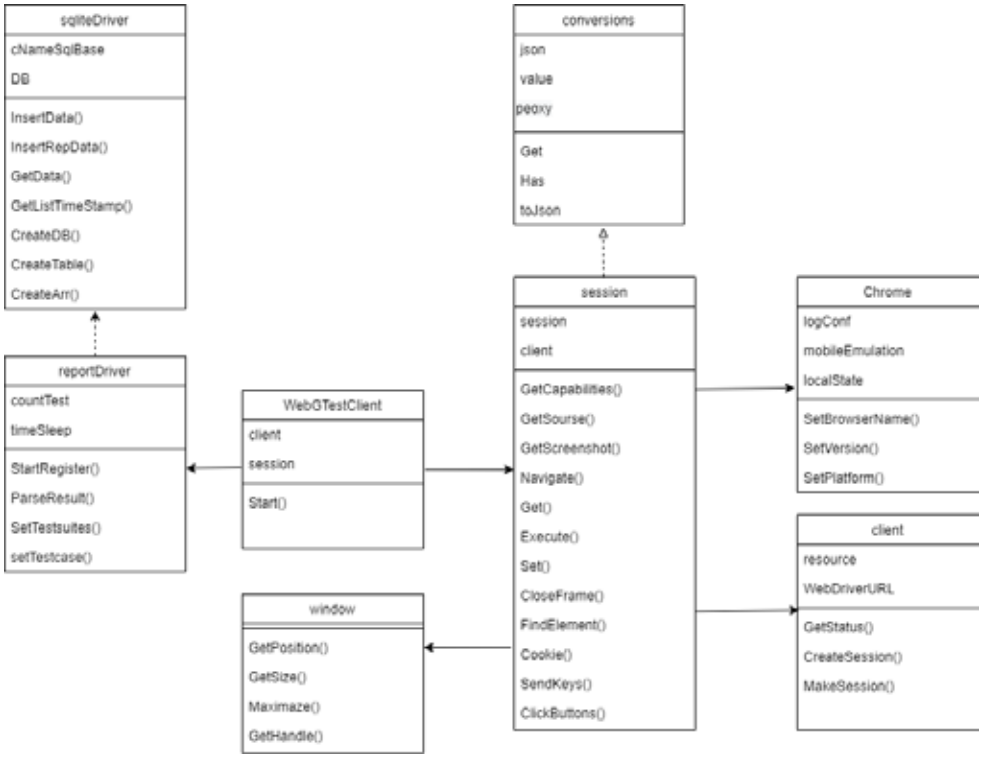

Рис. 2. Діаграма класів інформаційної системи

В процесі проектування програмної системи було визначено 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.

Спроектована система має спеціалізоване призначення, яке дозволяє взаємодіяти з web-сторінками, та проводити автоматизоване тестування функціональних можливостей. Таке рішення є актуальною задачею для тестувальників-автоматизаторів, та потребує використання спеціалізованого програмного оточення. До такого оточення слід віднести функціональний сервер Selenium, який дозволяє проводити тестування на хості сайта чи додатка. Однак, Selenium не є досить гнучким рішенням для проведення тестування. Він орієнтований для використання виключно web-мов програмування, що суттєвим чином зменшує ефективність тестування. Автоматизовані та функціональні тестові сценарії ноебхідно розробляти виключно використовуючи мову програмування C++, оскільки вона має більший вплив на взаємодію зі структурними та функціональними компонентами.

Отже, для взаємодії мови програмування C++ 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 C++ та їх взаємодії з спеціалізованим набором скриптів. Далі тестовий сценрій 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.

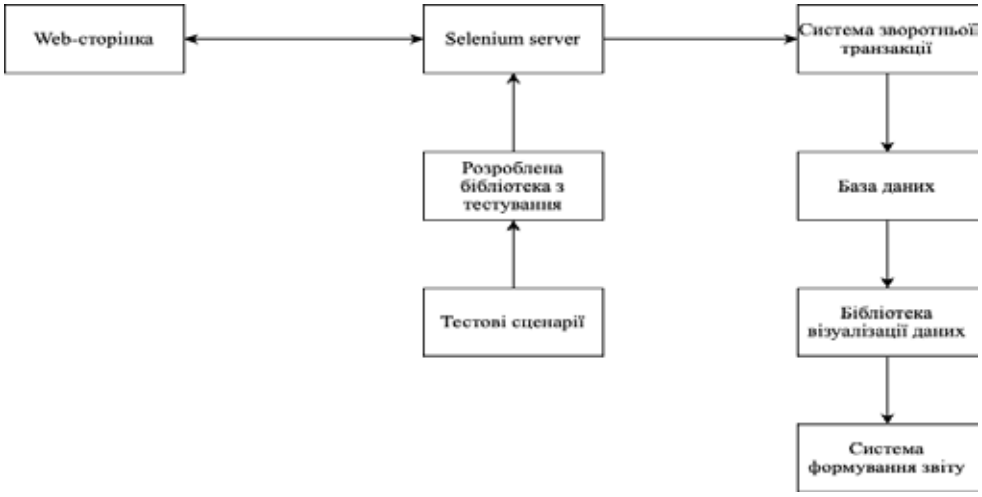

Рис. 3. Схема взаємодії спроектованих системних компонентів

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 трансює отримані результати до бази даних, після чого використовується бібліотека візуалізації графіки, яка будує діаграми та дозволяє структурувати інформацію в файлі. На рис. 3 зображено схему взаємодії системних компонентів в розробленій інформаційній системі.

Висновки та перспективи подальших досліджень. Результатом роботи є спроектоване спеціалізоване програмне оточення, яке включає в себе окрему функціональну бібліотеку та набір спеціалізованих автоматизованих тестових сценаріїв, які виконуються у взаємодії з Selenium Server. Також спроектовано гнучку та ефективну систему звітування за результатами тестів. На підставі аналізу структури традиційного звіту про дефекти програмних продуктів, які були виявлені при тестуванні, створена модифікована структура звіту про помилки. Запропонована структура звіту дозволяє не тільки поліпшити взаємодію розробників та тестувальників програмного забезпечення, а й використовується як засіб контролю за їх діяльністю. При цьому кожне поле звіту розглядається з точки зору його обліку для оцінки ефективності тестування. Розроблено методику кількісної оцінки ефективності тестування програмних продуктів на основі баг-репортів. Ефективність роботи інженера з оцінки якості оцінюється за формулою, яка отримана евристичним шляхом і враховує кількість і критичність помилок. Встановлено мінімальний достатній рівень ефективності тестування при використанні зазначеної методики. Надалі ефективним розвитком системи є впровадження самостійного платформного рішення з додання нових тестових сценаріїв та їх використанні при багаторазових перевірках ресурсів.

Список використаних джерел:

1. Принципи Software Testing Life Cycle (STLC). URL: https://softwaretestingfundamentals.com/software-testing-life-cycle.

2. Блєк Р. Ключевые процессы тестирования. Планирование, подготовка, проведение, совершенствование. М.: Лори. 2016. 537 с.

3. Эдгрен Р. The little black book on test design. NY : CreateSpace. 2011. 228 с.

4. Фаулер М., Райс Д., Фоммел М. Архитектура корпоративных программных приложений. М. : Вильямс. 2006. 544 с.

5. Бейзер Б. Тестирование черного ящика. Технологии функционального тестирования программного обеспечения и систем. М. : Вильямс. 2004. 320 с.

6. Криспин Л., Грегори Д. Agile-тестирование. Обучающий курс для всей команды. М. : Манн Иванов и Фербер. 2019. 528 с.

7. The ROI of Test Automation», Michael Kelly. URL: http://www.sqetraining.com/sites/default/files/articles/XDD8502filelistfilename1_0.pdf
8. Гленфорд М., Майерс Г., Баджетт Т., Сандлер К. Искусство тестирования программ. М. : Вильямс. 2012. 272 с.
9. Баранов С. Процесс разработки программных изделий. М. : ФИЗМАТЛИТ. 2000. 176 с.
10. Дастин Є. Автоматизоване тестування програмного забезпечення. впровадження, управління та експлуатація. М. : Лори. 2016. 592 с.
11. Джез Х., Хамбл Д., Фарли Д. Непрерывное развертывание ПО. Автоматизация процессов сборки, тестирования и внедрения новых версий программ. М. : Вильямс. 2011. 432 с.
12. Гленфорд М., Майерс Г., Баджетт Т. Искусство тестирования программ. М. : Вильямс, 2012. 272 с.
13. Мослей Д. Just Enough Software Test Automation. NY : CreateSpace. 2002. 260 с.
14. The Selenium Browser Automation Project documentation. URL: https://www.selenium.dev/documentation/en
15. Generic opensource Robot Framework for python. URL: https://robotframework.org/#introduction

References:

1. Principles of Software Testing Life Cycle (STLC). Access mode: https://softwaretestingfundamentals.com/software-testing-life-cycle. [in Ukrainian].
2. Blek, R. (2016). Key testing processes. Planning, preparation, implementation, improvement. M.: Laurie, 537 p. [in Russian].
3. Edgren, R. (2011). The little black book on test design. NY: CreateSpace, 228 p. [in English].
4. Fowler, M., Rice, D., Fommel, M. (2006). Architecture of corporate software applications. M.: Williams, 544 p. [in Russian].
5. Beizer, B. (2004). Black box testing. Functional testing technologies for software and systems. M.: Williams, 320 p. [in Russian].
6. Crispin, L., Gregory, D. (2019). Agile testing. Training course for the whole team. M.: Mann Ivanov and Ferber, 528 p. [in Russian].
7. The ROI of Test Automation, Michael Kelly. Access mode: http://www.sqetraining.com/sites/default/files/articles/XDD8502filelistfilename1_0.pdf [in English].
8. Glenford, M., Myers, G., Budgett, T., Sandler, K. (2012). The Art of Software Testing. M.: Williams, 272 p. [in Russian].
9. Baranov, S. (2000). Process of software products development. M.: FIZMATLIT, 176 p. [in Russian].
10. Dustin, Є. (2016). Automated software testing. vprovadzhennya, management and exploitation. M.: Lori, 592 p. [in Ukrainian].
11. Jez, H., Humble, D., Farley, D. (2011). Continuous software deployment. Automation of assembly, testing and implementation of new versions of programs. M.: Williams, 432 p. [in Russian].
12. Glenford, M., Myers, G., Budgett, T. (2012). The Art of Software Testing. M.: Williams, 272 p. [in Russian].
13. Mosley, D. (2002). Just Enough Software Test Automation. NY: CreateSpace, 260 p. [in English].
14. The Selenium Browser Automation Project documentation. Access mode: https://www.selenium.dev/documentation/en [in English].
15. Generic opensource Robot Framework for python. Access mode: https://robotframework.org/#introduction [in English].