УДК 004.9 DOi 10.30837/ bi.2025.1(102).07

С.Г. Удовенко[1], В.А. Затхей[2], О.В. Тесленко[3], Л.Е. Чала[4]

[1]ХНЕУ ім. С. Кузнеця, м. Харків, Україна, serhiy.udovenko@hneu.net,
ORCID iD: 0000-0001-5945-8647

[2]ХНЕУ ім. С. Кузнеця, м. Харків, Україна, zathey_va@ukr.net,
ORCID iD: 0000-0003-4426-7789

[3]ХНЕУ ім. С. Кузнеця, м. Харків, Україна, oleh.teslenko@hneu.net,
ORCID iD: 0000-0003-3105-9323

[4]ХНУРЕ, м. Харків, Україна, larysa.chala@nure.ua,
ORCID iD: 0000-0002-9890-4790

АВТОМАТИЗОВАНЕ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ МОДУЛІВ ІНФОРМАЦІЙНИХ СИСТЕМ

Досліджено проблему автоматизованого тестування програмного забезпечення (ПЗ) модулів інформаційних систем (ІС) на прикладі тестування веб-застосунків. Розглянуто схему генерації тестових сценаріїв для тестування ПЗ модулів ІС (на прикладі веб-сайтів визначеної предметної області). Запропоновано технологію оптимізації процесів тестування веб-сайтів з використанням спеціалізованого тестового фрєймворку та інструментів системи керування контентом Sitecore CMS. Розглянуто можливість застосування в запропонованій системі тестування концепції безперервної інтеграції та доставки (CI/CD), що дозволяє запускати різні типи тестів на кожному етапі тестування і завершувати їх розгортанням тестованого коду в кінцевий продукт. Працездатність запропонованої технології автоматизованого тестування модулів ІС досліджено на прикладі тестування веб-застосунків. Визначено, що використання цієї технології дозволяє скоротити витрати часу на тестування веб-додатків.

АВТОМАТИЗОВАНЕ ТЕСТУВАННЯ, ВЕБ-ЗАСТОСУНОК, ФРЕЙМВОРК, ТЕСТОВИЙ СЦЕНАРІЙ, ІНФОРМАЦІЙНА СИСТЕМА, ТЕСТОВА МОДЕЛЬ

S.G. Udovenko, V.A. Zathey, O.V. Teslenko, L.E. Chala. Automated software testing of information systems modules. The paper explores the problem of automated software testing (SW) of information system (IS) modules, using web application testing as a case study. A scheme for generating test scenarios for IS module testing is presented, specifically applied to websites within a defined subject area. The paper proposes a technology for optimizing website testing processes using a specialized test framework and Sitecore CMS tools. The integration of continuous integration and delivery (CI/CD) concepts into the testing system is examined, enabling the execution of various types of tests at different testing stages, culminating in the deployment of the tested code as a final product. The performance of the proposed automated testing technology is evaluated through web application testing, demonstrating a significant reduction in the time required for testing processes.

AUTOMATED TESTING, WEB APPLICATION, FRAMEWORK, TEST SCENARIO, INFORMATION SYSTEM, TEST MODEL

Вступ

Функціональні модулі інформаційної системи (ІС) – це взаємопов'язані частини системи, кожна з яких виконує певні функції, необхідні для досягнення загальної мети системи. Вони складаються з елементів, які визначають функціональні можливості системи та забезпечують її роботу [1].

Існує широка градація модулів ІС (МІС), що, насамперед, залежить від функціонального призначення інформаційних систем та їх структурних особливостей. Втім, такі модулі, як функціональні підсистеми та спеціалізовані веб-додатки (зокрема веб-сайти структурних одиниць ІС) є притаманними для переважної більшості розроблюваних ІС.

Це обумовлює важливість уважного ставлення до створення та удосконалення ефективних технологій побудови веб-додатків, як важливих модулів ІС.

Запорукою ефективності створюваних модулів ІС є розроблення працездатного програмного забезпечення (ПЗ МІС). Важливим етапом розробки програмного забезпечення є етап тестування. Складність і різноманіття функцій сучасних програм вимагає застосування спеціальних методів аналізу стану і працездатності програмного забезпечення протягом усього циклу розробки, впровадження та супроводу. У широкому сенсі, тестування – це одна з технік контролю якості, яка включає планування, складання тестів, безпосереднє виконання тестування та аналіз отриманих результатів. На сьогодні забезпечення якості програмних продуктів, що контролюється на етапі тестування, є одним з провідних напрямів в ІТ-індустрії.

До найбільш важливих критеріїв вибору технології створення та тестування веб-сайтів слід віднести: розмір та тип проекту; складність проекту; швидкість розробки; вартість послуг спеціалістів; наявність доступних інструментів для розробки; гнучкість рішення; тенденції розвитку проекту; наявність детальної

документації; вартість підтримки; кросплатформеність; можливість інтеграції з іншими рішеннями тощо [2, 3].

Автоматизація процесів тестування представляє особливий інтерес, тому що дозволяє значно скоротити час проходження системою повного комплексу тестових сценаріїв та мінімізувати при цьому кількість трудових витрат на цей процес. Автоматизовані системи тестування вже стали невід'ємною частиною процесів розробки модулів ІС. Особливу увагу слід приділити сфері веб-розробок, що стрімко розвивається та стає все більш поширеною областю в інженерії програмного забезпечення.

В області веб-розробок на сьогоднішній день існує велика кількість різних систем автоматизації тестування, що відрізняються функціональними можливостями, ступенем стандартизації, опрацюванням призначеного для користувача інтерфейсу, правилами ліцензування [4, 5].

Незважаючи на те, що в області тестування вебзастосунків (зокрема, веб-сайтів МІС) здійснюється велика кількість теоретичних та практичних досліджень, в ній досі залишається чимало важливих питань, що потребують вирішення [6]. Одним з них є побудова систем автоматичного тестування вебсайтів та генерація тестових сценаріїв для них. Таким чином, розроблення ефективної технології генерації тестових сценаріїв та універсальних шаблонів для автоматичного тестування веб-сайтів є актуальним завданням.

Слід також зазначити, що на сьогодні потужними засобами автоматизованого тестування створюваних веб-сайтів є застосування фреймворків (середовищ розробки для програмістів з готовими шаблонами та інструментами) та систем керування контентом CMS (Content Management System), призначених для організації веб-сайтів чи інших інформаційних ресурсів в Інтернеті чи окремих комп'ютерних мережах. В загальному випадку CMS використовується як готове рішення, в якому потрібно лише налаштувати контент, проте на основі фреймворку можна створити власну CMS під специфіку та функціональність проекту. Одною з таких систем є Sitecore CMS. Sitecore CMS використовує платформу ASP.NET і працює під управлінням IIS (Internet Information Services), що відкриває чимало можливостей для розробки вебзастосунків.

Метою цієї статті є розроблення та дослідження технології оптимізації процесів автоматизованого тестування веб-сайтів з використанням запропонованого тестового фрєймворку, засобів Sitecore CMS та генерації тестових сценаріїв.

Реалізація такої технології дозволить автоматизувати процедуру функціонального тестування розроблюваних веб-застосунків ІС та скоротити сумарні витрати на реалізацію життєвого циклу їх проеєктування.

Відповідно до поставленої мети вирішуються такі завдання:

– аналіз існуючих технологій автоматизації модульного тестування веб-застосунків інформаційної системи;

– розроблення схеми генерації тестових сценаріїв для автоматичного тестування ПЗ модулів ІС (на прикладі веб-сайтів визначеної предметної області);

– розроблення технології оптимізації процесів тестування веб-сайтів з використанням пропонованого тестового фрєймворку Sitetest та засобів Sitecore;

– експериментальне дослідження запропонованого підходу.

1. Технології автоматизації тестування

Функціональне тестування модуля інформаційної системи – це процес перевірки того, як програмне забезпечення модуля виконує певні функції, порівнюючи його поведінку з вимогами. Мета такого тестування – переконатися, що модуль системи працює відповідно до специфікацій.

До основних типів фунціоналного тестування слід, зокрема, віднести модульне тестування (перевірка окремих модулів або компонентів ІС) та інтеграційне тестування (перевірка взаємодії між різними модулямиІС).

Модульне та інтеграційне тестування зазвичай автоматизуються, що дозволяє значно прискорити загальний процес тестування і знизити ризик помилок [7, 8].

У контексті тестування ПЗ модулів ІС функціональне автоматизоване тестування може включати: перевірку введення та обробки даних; перевірку правильності відображення інформації; тестування роботи з інтерфейсом користувача; перевірку обробки різних сценаріїв користувачів; тестування безпеки та авторизації.

На сьогодні ієрархію технологій автоматизованого тестування (від вищого рівня до нижнього) модулів ІС можно умовно відобразити наступним чином [9]:

– тестування під управлінням поведінкою;

– тестування під управлінням ключовими словами;

– тестування під управлінням даними;

– запис та відтворення;

– використання фреймворків.

Кожна з цих технологій автоматизації тестування, в свою чергу, базується на відповідному наборі технічних рішень (інструментальних засобів, мов програмування, способів взаємодії з тестованим за стосунком тощо) та має свої переваги та недоліки.

Більшість сучасних технологій автоматизації тестування максимально сфокусовані на технічних

аспектах поведінки тестованих модулів та мають загальний недолік: з їх допомогою складно перевіряти високорівневі сценарії, що призначені для користувача (а саме в них і зацікавлені замовники і користувачі). Цей недолік частково виправляє тестування під керуванням поведінкою (Behavior-driven development, BDD), де акцент робиться не на окремих технічних деталях, а на загальній працездатності ПЗ модуля при вирішенні типових користувальницьких завдань.

Такий підхід не тільки спрощує виконання цілого класу перевірок, а й полегшує взаємодію між розробниками, тестувальниками, бізнес-аналітиками і замовником, тому що в основі підходу лежить достатньо проста формула «given-when-then»:

– given («за умови»): описує початкову ситуацію, в якій знаходиться користувач в контексті роботи з тестованим застосунком;

– when («якщо»): описує набір дій користувача в даній ситуації;

– then («тоді»): описує очікувану поведінку програми модуля ІС.

Такий принцип опису перевірок дозволяє навіть учасникам проекту, що не мають спеціальної підготовки, брати участь в розробці і аналізі тест-кейсів, а для фахівців з автоматизації спрощується створення коду автоматизованих тест-кейсів, тому що така форма є стандартною, єдиною і при цьому надає достатньо інформації для написання високорівневих тест-кейсів. Існують спеціальні технічні рішення (наприклад, Behat, JBehave, NBehave, Cucumber), що спрощують реалізацію тестування під керуванням поведінкою.

Недоліки тестування під управлінням поведінкою: високорівневі поведінкові тест-кейси пропускають багато деталей, а тому можуть не виявити частину проблем в застосунку або не надати необхідної для розуміння виявленої проблеми інформації.

Тестування під управлінням ключовими словами є логічним розвитком ідеї про винесення зовні тест-кейса даних та команд (опису дій). Прикладом інструментального засобу автоматизації тестування під керуванням ключовими словами є фреймворк Selenium IDE, де кожна операція тест-кейса описується в вигляді: дія (ключове слово) – необов'язковий параметр 1 – необов'язковий параметр 2.

Переваги та недоліки тестування під управлінням ключовими словами:

– переваги: максимальне усунення надмірності коду тест-кейсів; можливість побудови міні-фреймворків, які вирішують широкої спектр завдань; підвищення рівня абстракції тест-кейсів і можливість їх адаптації для роботи з різними технічними рішеннями; зручне зберігання і зрозумілий для користувачів формат даних і команд тест- кейса; можливість повторного використання коду тест-кейсів для вирішення нових завдань; можливість розширення (додавання нової поведінки тест-кейса на основі вже реалізованого варіанту);

– недоліки: загальна складність розробки; висока ймовірність наявності помилок в коді тест-кейса; висока складність виконання низькорівневих операцій, якщо код тест-кейса не підтримує відповідні команди; ефект від використання даного підходу настає далеко не відразу (спочатку йде тривалий період розробки та налагодження тест-кейсів і допоміжної функціональності); необхідність застосування мови ключових слів, що використовуються в тест-кейсах.

Тестування під управлінням даними (Data Driven Testing, DDT) є підхідом до створення автоматизованих тестів, при якому тест має приймати набір вхідних параметрів, а еталонний стан, з яким він повинен порівняти результат, отримується під час прогонки вхідних параметрів.

До типових прикладів використання тестування під керуванням даними відносяться: перевірка авторизації і прав доступу на великому наборі імен користувачів і паролів; багаторазове заповнення полів форм різними даними і перевірка реакції модуля; виконання тест-кейса на основі даних, отриманих за допомогою комбінаторних технік.

Технологія запису і відтворення (Record & Playback) стала актуальною з появою достатньо складних засобів автоматизації, що забезпечують глибоку взаємодію з тестованим застосунком і операційною системою. Використання цієї технології, зазвичай, зводиться до наступних основних кроків: тестувальник самостійно виконує тест-кейс, а засіб автоматизації записує його дії; результати записів формуються у вигляді коду на високорівневій мові програмування; тестувальник редагує отриманий код; готовий код автоматизованого тест-кейса виконується для проведення тестування в автоматизованому режимі.

Ця технологія при високої складності внутрішньої реалізації є простою у використанні, тому часто застосовується для попереднього навчання фахівців з автоматизації тестування. Її основні переваги та недоліки:

– переваги: простота освоєння тестувальниками; швидке створення «скелета» тест-кейса за рахунок запису ключових дій з тестованим застосунком; автоматичний збір технічних даних про тестований застосунок (ідентифікаторів і локаторів елементів, написів, назв тощо); автоматизація рутинних дій (заповнення полів, натискань на посилання, кнопки);

– недоліки: лінійність тест-кейсів (в записі не буде циклів, умов, викликів функцій і інших характерних для програмування і автоматизації функцій); запис зайвих дій (як помилкових випадкових дій тестувальника з тестованим застосунком, так і перемикань на інші застосунки та роботу з ними); незручні імена

змінних, незручне оформлення коду тесткейса, відсутність коментарів, що ускладнюють підтримку і супровід тест-кейса в подальшому; низька надійність самих тест-кейсів через відсутність перевірки умов.

Фреймворки автоматизації тестування зазвичай є успішно розвиненими рішеннями, що поєднують в собі переваги тестування під керуванням даними та ключовими словами, а також можливість реалізації додаткових рішень на високому рівні абстракції.

Фреймворків автоматизації тестування можуть суттєво різнитися, але їх об'єднує кілька спільних рис, а саме: висока абстракція коду (немає необхідності опису кожної елементарної дії) зі збереженням можливості виконання низькорівневих дій; універсальність використовуваних підходів; досить високуа якість реалізації (для популярних фреймворків).

Зазвичай кожен фреймворк спеціалізується на конкретних видах, рівнях та технологіях тестування. Існують фреймворки для модульного тестування (наприклад, сімейство xUnit), тестування веб-застосунків (наприклад, сімейство Selenium), тестування мобільних додатків, тестування продуктивності тощо. Існують вузько і широко спеціалізовані, а також безкоштовні і платні фреймворки, що оформлені як бібліотеки на деякій мові програмування або як додатки з графічним інтерфейсом.

Основні переваги та недоліки тестування модулів ІС з використанням фреймворків:

– переваги: широке розповсюдження; універсальність в рамках обраного набору технологій; високий рівень абстракції; наявність набору готових рішень і описів відповідних кращих практик застосування того чи іншого фреймворка для вирішення конкретних завдань;

– недоліки: необхідність витрат часу на вивчення фреймворка тестувальниками; висока ймовірність необхідності модифікації існуючого або створення власного фреймворка для вирішення конкретних завдань; виникнення додаткових труднощів при переході на інший фреймворк; високий ризик вибору невідповідного фреймворка.

Існують певні напрями оптимізації тестування, метою яких є: прискорення виконання тестування; вивільнення людських ресурсів; збільшення тестового покриття; виключення людського фактору; надання можливості переглядати звіт з проходження тесткейсів.

Їх можна, зокрема, реалізувати за допомогою автоматизації тестування на проєкті. При цьому мають бути протестованими: поведінка програмного продукту з точки зору контентного менеджера (додавання, редагування та видалення сторінок, компонентів тощо) та з точки зору кінцевого користувача (правильність відображення компонентів стосовно співпадіння даних, які заповнив контент-менеджер, з даними, що відображаються), а також візуальна відповідність компонент результатів вимогам. Інколи для оптимізації цих сфер тестування необхідно розробити фреймворк для UI тестування (перевірка правильності роботи компонентів з точки зору користувача). Для перевірки роботи програмного продукту з точки зору контент-менеджера достатнім є його тестування за допомогою API.

Розглянемо особливості вибору інструментів автоматизованого тестування модулів ІС. Згідно з метою статті необхідно проаналізувати існуючі тестові фреймворки, що найбільш поширено використовуються для оптимізації процесів тестування веб-сайтів ІС.

До таких фреймворків слід, насмперед, віднести інструмент Selenium з відкритим вихідним кодом, призначений для автоматизації тестування веб-додатків.

Фреймворк Selenium підтримується операційними системами Windows, Mac, Linux, а також браузерами Chrome, Firefox, IE і Headless. Скрипти для даного фрейма можна написати на більшості популярних сьогодні мов програмування: Java, Groovy, Python, C #, PHP, Ruby і Perl [10, 11].

Фреймворк Katalon Studio також є достатньо ефективним інструментом для автоматизації процесу тестування веб-додатків, мобільних додатків і вебсервісів. Він перейняв у Selenium переваги, пов'язані з інтегрованою автоматизацією тестування ПЗ.

Фреймворк Unified Functional Testing (UFT) є популярним комерційним інструментом для функціонального тестування. Він надає повний набір функцій для тестування API, веб-сервісів, а також для тестування графічного інтерфейсу десктопних, мобільних і веб-додатків на всіх існуючих платформах. Для даного інструменту передбачена розширена функція розпізнавання об'єктів на основі зображень, багаторазові тестові компоненти і документація по автоматичному тестуванню.

Фреймворк Watir – це інструмент з відкритим вихідним кодом для автоматизації тестування вебзастосунків, що використовує бібліотеки Ruby. Для Watir передбачена можливість крос-браузерного тестування для браузерів Firefox, Opera, headless-браузерів і IE. Він також підтримує кероване даними тестування і інтегрований з такими інструментами BBD, як RSpec, Cucumber і Test / Unit.

Фреймворк TestComplete є інструментом для тестування десктопних, мобільних і веб-додатків. TestComplete підтримує такі мови сценаріїв, як JavaScript, VBScript, Python і C ++ Script. За допомогою TestComplete тестувальники можуть виконувати тестування з використанням ключових слів і кероване даними тестування. В інструменті також передбачена зручна функція запису і відтворення процесу тестування.

На основі розглянутих фреймворків може бути створена система керування контентом CMS, призначена для автоматизованого тестування веб-сайтів. Серед існуючих систем цього типу в останні роки набирає значної популярності система Sitecore CMS. Система Sitecore CMS володіє надійним набором можливостей для автоматизованого тестування, персоналізації та оптимізації , що робить її ідеальною платформою для тестування та уточнення створюваних веб-застосунків [12]. Так як Sitecore використовує платформу ASP.NET, то для кращої інтеграції з проектом написання модифікованого тестового фреймворку доцільно використовувати мову програмування C# і відповідно платформу .NET і середовище розробки Microsoft Visual Studio 2017.

Розглянемо переваги платформи ASP.NET, що використовуються в системі Sitecore CMS: платформа .NET ґрунтується на єдиній обєктно-орієнтованій моделі; до складу платформи .NET входить інструмент захисту програми тестування від втрат памяті і від необхідності звільняти ресурси; будь-яка програма, розроблена з допомогою .NET є автономною, в тому сенсі, що не залежить від інших програм та від ОС; встановлення програми може бути проведене звичайним копіюванням файлів; в платформі .NET використовуються безпечні типіи даних, що підвищує надійність програм та їх сумісність; програма взаємодіє з єдиною моделлю обробки помилок; всі помилки оброблюються механізмом виключних ситуацій, що дозволяє запобігти неоднозначностям [13].

Для підвищення можливостей автоматичного тестування ПЗ модулів ІС останнім часом застосовуються методи генерації тестових сценаріїв (на прикладі веб-сайтів визначеної предметної області) та створення відповідних універсальних шаблонів.

Таким чином, можна зробити висновок про доцільність побудови фреймворку автоматизованого тестування на основі оптимізації процесів тестування веб-сайтів з використанням засобів Sitecore CMS та схеми генерації тестових сценаріїв.

2. Схема генерації тестових сценаріїв для автоматичного тестування веб-застосунків визначеної предметної області

До важливих завдань автоматизованого тестування ПЗ модулів ІС слід віднести побудову тестових сценаріїв, що брали б до уваги специфічні вимоги до тестованих веб-застосунків та озволяли б ефективно виявляти помилки в розроблених кодах. Такі тестові сценарії можуть бути створені на основі моделей веб-застосунків [14]. Послідовність їх створення наведено на рис. 1.

Рис. 1. Генерація сценаріїв тестування на основі моделі веб-застосунку

Модель веб-застосунків дає можливість генерувати набори варіантів сценаріїв тестування, де визначаються умови, вхідні дані, очікувані результати роботи системи та оцінювання результатів тестування з використанням компаратора. Порівняння в компараторі очікуваних результатів та поточних спостережень дозволяє зробити висновок щодо наявності помилок в тестованому модулі та його відповідності до вимог. В разі необхідності можна модифікувати існуючу модель або ініціювати додаткову генерацію варіантів тестування.

Особливості автоматизованого тестування у веб-проєктах обумовлюється наявністю коротких циклів розробки, розподіленою архітектурою, кросплатформністю та кросбраузерністю.

Для моделювання та подальшого тестування складних веб-застосунків доцільно використовувати ієрархічний підхід.

Відзначимо, що процес формування тестових сценаріїв є індивідуальним для кожного конкретного проєкту та може відбуватися паралельно з розробленням ПЗ тестованого модуля. Достатньо специфічною особливістю проєктів створення веб-застосунків є неодноразове повторення типових процедур, які мають лише незначні функціональні відмінності. До таких проєктів можна віднести розробку інтернет-магазинів, блогів, порталів новин тощо. Звичайно, що за таких умов розробка тестових сценаріїв для кожного окремого проєкту є недоцільною та раціональному використанню ресурсів.

Пропонований підхід передбачає побудову ієрархічної структури універсальних тестових сценаріїв. Верхній рівень цієї структури містить універсальні тестові сценарії, що характерні для досліджуваної предметної області. Найнижчий рівень ієрархічної структури відповідає області взаємодії з елементами візуального інтерфейсу конкретного проєкту. Адаптація розроблених шаблонних сценаріїв до конктетних проєкт івздійснюється саме на цьому

ієрархічному рівні. Ієрархічна структура тестових сценаріїв дозволяє здійснювати тестування на основі моделей, де враховуються функціональні вимоги до веб-додатків обраної типової предметної області та налаштування для конкретної реалізації.

Після налаштування моделі під конкретний веб-застосунок остаточно генеруються тестові набори.

Узагальнена ієрархічна структура успадкованих тестових сценаріїв містить ядро та змінну частину. До ядра вносять опис тестових випадків, що залишається незмінним від проекту до проекту в рамках певної предметної області. Змінна частина структури (словник проєкту) місить відсилання до існуючих точок входу проєкту (наприклад, адреси сторінок, посилання на активні елементи інтерфейсу користувача, посилання на блоки певного призначення).

Загальну структуру системи автоматичного тестування з використанням набору шаблонів наведено на рис. 2.

Ядро системи тестування в свою чергу поділяється на набір шаблонів тестових сценаріїв та бібліотеку операторів. Такий поділ є корисний для виключення дублювання коду в разі повторення однакових кроків для різних тестових сценаріїв. Під тестовим сценарієм будемо розуміти сукупність тестових випадків, які ініціюють перевірку працездатності окремого фрагменту модуля, що тестується. Тестові сценарії зберігаються в окремих файлах і є автономними функціональними одиницями. Вміст та характеристики тестових сценаріїв відповідають конкретній предметної області.

Рис. 2. Структура системи автоматичного тестування з використанням шаблонів

Шаблони тестових сценаріїв обєднують сукупності тестових випадків, які визначають послідовність дій потенційного користувача, що впливає певним чином на стан тестованого модуля системи та конкретний очікуваний результат. Перевірка працездатності тестованого модуля відбувається за результатами порівняння очікуваного результату з реальним.

Система автоматичного тестування з використанням шаблонів має забезпечувати:

– універсальність шаблонів, що визначають повноту покриття тестовими випадками предметної області (тобто тестових сценаріїв ядра);

– мінімізацію кількості елементів словника проєкту, що визначає складість формування словників для нових проєктів.

Бібліотека операторів містить сукупність повторюваних дій, що можуть використовуватися в різних тестових сценаріях. Такі оператори взаємодіють з даними зі словника проекту для отримання доступу до необхідних елементів інтерфейсу користувача веб-застосунку. Всі оператори поділяються на оператори переходу, оператори дії та оператори перевірки.

Оператори переходу використовуються для переміщення між сторінками веб-застосунку і мають забезпечувати імітацію дій потенційних користувачів. Система автоматичного тестування має використувати лише ті елементи управління, що доступні реальним користувачам. Кожен тестовий випадок починається з виклику оператора переходу до розділу системи, розглядається цей випадок.

Оператори дії визначають варіанти взаємодії з елементами управління інтерфейсу користувачів (наприклад, з кліками по активним елементам, введенням даних з клавіатури тощо);

Оператори перевірки реалізують різні методи перевірки стану тестованого модуля. Такі методи перевіряють, зокрема, наявність на сторінках певних елементів та вміст текстових фрагментів, а також порівнюють поточні значення параметрів системи з раніше збереженими значеннями.

Словник проекту використовується для модифікації системи автоматичного тестування під потреби конкретних проєктів. Він містить специфічні для тестованого веб-застосунку дані, а саме: найменування елементів графічного інтерфейсу та посилання на них; відклики на результати виконання операцій; опис структур даних аналізованих об'єктів тощо. В цілому складові частини словнику проєкту можуть бути поділені на такі групи:

– ідентифікатори елементів інтерфейсу користувача, що використовують CSS-селектори;

– текстові дані та повідомлення, характерні для різних тестових випадків (за їх наявністю тестова система тестування визначає успішність виконання тестового випадку);

– тригери (параметри, що визначають алгоритми проведення тестів та характеризуються булевими значеннями для тестових випадків проєкту).

Метод побудови та використання ієрархічної структури шаблонів автоматизованого тестування

(для конкретної предметної області) передбачає реалізацію наступних кроків:

Крок 1. Аналіз предметної області та визначення основних функціональних вимог до для тестованого веб-застосунку.

Крок 2. Формування ядра системи тестування, де міститься опис універсальних тестових сценаріїв з урахванням визначених функціональні вимоги, а також бібліотека операторів, що визначають сукупність повторюваних дій.

Крок 3. Формування словника (змінної частини для конкретного веб-застосунку, де містяться адреси веб-сторінок, ідентифікатори елементів інтерфейсу користувачів, параметри тестованого веб-застосунку тощо.

Крок 4. Інтеграція розроблених шаблонів до системи автоматичного тестування.

Крок 5. Модифікація словника для конкретного тестованого веб-застосунку.

Крок 6. Тестування веб-застосунку з використанням шаблонів кінцевих тестових сценаріїв.

Основними вимогами до програмної реалізації запропонованого методу є: наявність гнучкої системи команд і операторів; можливість автоматичної адаптації до зміни умов тестування з використанням циклів і розгалужень; підтримка ієрархічних структур і успадкування; підтримка оголошень функцій і базової можливості повторного використання коду.

Таким чином, шаблони тестування мають бути зручними для користувачів та тестувальників.

В роботі [14] був дослідженний варіант програмної реалізації запропонованого методу з використанням фреймворку Selenium. Втім такий варіант характеризується наявністю певних недоліків, повязаних з недосконалістю інтерфейсу записи сценаріїв в порівнянні з комерційними платними аналогами, а також з відсутністю вбудованої системи формування звітів за результатами автоматизованого тестування.

Є доцільним розглянути можливість реалізації методу автоматичного тестування на основі використання шаблонів, що базується на застосуванні пропонованого тестового фрєймворку Sitetest та засобів Sitecore CMS.

3. Архітектура тестового фрєймворку Sitetest для автоматизації тестування веб-сайтів

Розглянуто можливість оптимізації процесів тестування веб-сайтів з використанням спеціалізованого тестового фрєймворку та засобів Sitecore CMS.

Для створення тестового фреймворку було обрано наступний набір інструментів та мови програмування: мова програмування C#, середовище розробки Visual Studio 2017, інструмент для UI тестування та взаємодії з веб елементами Selenium, BDD інструмент SpecFlow, Sitecore API бібліотека для можливості тестування функціональності з участю контент-менеджера.

Побудова фреймворків для тестування модулів ІС (зокрема, тестування веб-застосунків) має враховувати загальні принципи розроблення інструментів автоматизованого тестування. Зокрема, згідно з принципом інверсії залежностей (dependency inversion principle) класи верхніх рівнів фреймворку тестування не повинні залежати від класів нижніх рівнів. Класи нижнього рівня реалізують базові операції (передачі даних по мережі, підключення до бази даних тощо).. Класи високого рівня містять більш складну бізнес-логіку програми, яка спирається на класи низького рівня для здійснення більш простих операцій. відбувається проектування. Відповідно до принципу інверсії залежностей слід розширювати низькорівневі класи і використовувати їх разом з класами бізнес-логіки, не змінюючи код останніх. В процесі реалізації тестового фреймворку цей принцип має використовуватися для створення шарів SpecFlow. В архітектурі тестового застосунку ці шари визначають зв'язок між .feature файлами та C# класами. Крім того для побудови остаточного варіанту фреймворку тестування треба спочатку створювати тестові сценарії без прив'язки до C# коду. Потім необхідно створити класи, де визначаються кроки тестування, а інші класи, (наприклад класи Page Object та Web Component) мають містити лише логіку роботи методів, що використовуються для роботи верхнього шару фреймворку [15].

На рис. 3 наведено архітектуру пропонованого фреймворку Sitetest, що використовується для тестування веб-сайтів.

Архітектуру тестового фреймворку Sitetest містить такі модулі та шари:

– ядро (Framework Core). Цей модуль задає конфігурацію веб-драйвера, розширення для веб- драйвера та веб-елементу, різні конфігураційні класи, класи з власними обгортками над очікуваннями тощо;

– шар SpecFlow Tests. Цей шар містить feature-файли та імплементацію кроків для них;

– тестова модель (TestModel). Цей модуль фреймворка містить декілька шарів. Перший шар – це шар сторінок (Page Objects), що допомагає інкапсулювати роботу з окремими елементами сторінки, дозволяє зменшити розмір коду та полегшити його підтримку (якщо, наприклад, дизайн однієї зі сторінок буде змінений, то потрібно буде переписати тільки відповідний клас, що описує цю сторінку). Другий шар є сервісний компонентом (Page Service), що містить класи, які відповідають за взаємодію з елементами сторінок, а також класи, які відповідають за перевірки;

– шар Sitecore, що відповідає за взаємодію з Sitecore API та використовується для тестування правильності функціонування додатку зі сторони контент- менеджера.

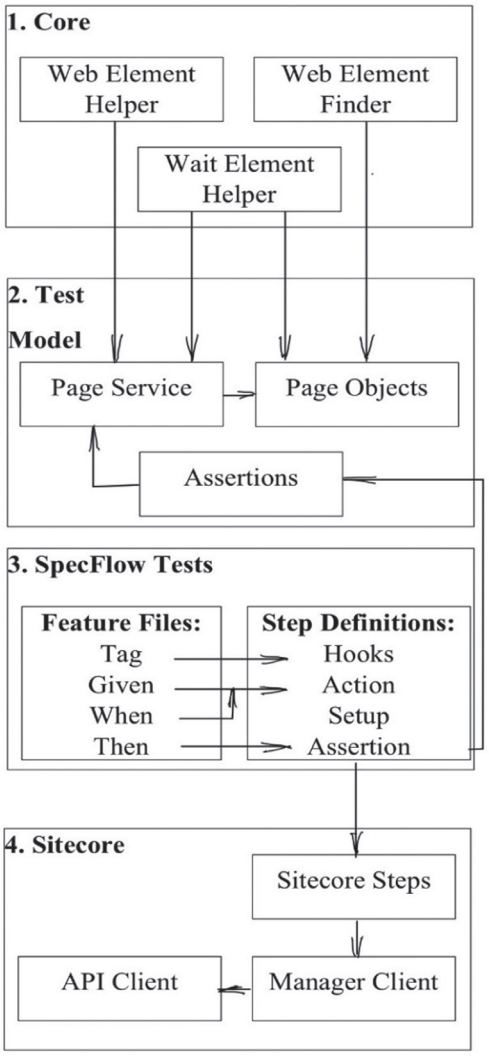

Рис. 3. Архітектура тестового фреймворку Sitetest для автоматизації тестування веб-сайтів

Функції та їх опис для шарів SpecFlow Tests, TestModel та Sitecore наведені в таблицях 1, 2, 3 відповідно.

Таблиця 1

Функції шару SpecFlow Tests

Функція	Опис функції
Extensions	Трансформація вхідних параметрів BDD сценаріїв до C# об'єктів та примітивних типів даних. Логіка порівняння вхідних параметрів сценаріїв.
Features	Тестові сценарії у вигляді послідовності кроків.
Step Definitions	Приведення BDD кроків до C# методів. Логіка порівняння очікуваного та актуального результатів. Логіка дій кінцевого користувача на веб-сайті.
App.config	Конфінурація тестового фреймворку. Змінні оточення. Глобальні тестові дані. Адреси веб-сторінок.

Тестові сценарії у вигляді послідовності кроків генеруються в шарі SpecFlow Tests на серверному тестовому стенді в Sitecore CMS (в цьому випадку запуск і збір результатів тестування здійснюється планувальником тестового стенда) або на локальній машині тестувальника (у такому випадку локальна машина є тестовим стендом і запуск тестових сценаріїв здійснюється з командного рядка в консолі ОС, а результат роботи також буде відображено в консолі).

Таблиця 2

Функції шару TestModel

Функція	Опис функції
Web	C# шаблони веб-сторінок та веб-компонентів. Веб-сторінки – композиція веб-компонентів.
Utils	Допоміжні класи для спільної логіки роботи із рзіними типами даних (string, DateTime, int). Логіка генерації випадкових даних.
Entity	Представлення веб-компонентів у вигляді C# об'єктів. Логіка порівняння C# об'єктів (Equals та HashCode методи).
TestContext	Кеш тестових даних, згенерованих протягом запуску тестів. Стратегія ініціалізації тестових даних та компонентів в залежності від обраного режиму запуску тестів (Mobile та Desktop).
UserFlow	Логіка реєстрації нового користувача в системі. Логіка активації нового акаунту.
Constants	Глобальні константи. Перерахування для параметризації BDD кроків.

Таблиця 3

Функції шару Sitecore

Функція	Опис функції
Core	Логіка перетворення API відповідей із Sitecore до C# об'єктів.
Client	API клієнт для взаємодії із Sitecore REST сервісом. Методи, що відповідні REST CRUD операціям.
Entity	Конкретні класи-шаблони Sitecore компонентів. Представлення узагальнених полей (кнопки, текстові блоки, багатовибірні поля).
Models	Спільні C# моделі для роботи із Sitecore API (логін, створення, оновлення, видалення).
Utils	Допоміжні класи для роботи із різними типами полей на Sitecore (текстові блоки, кнопки, випадаючі меню).
Data Holder	Кеш даних, зібраних із Sitecore протягом запуску тестів. Дані, що чекають на публікацію у master базі.
Default Data	Тестові дані для конфігурації базових Sitecore компонентів, спільні для кожного запуску тестів.

Використання шарів в архітектурі фреймворку надає процесу тестування такі переваги:

– спрощує розробку тестів за рахунок перевикористання бізнес-функцій та шаблонів, які повторюються практично в кожному тесті;

– спрощує підтримку тестів (якщо змінюється верстка сторінки або навігація, то зміни достатньо внести лише в кілька класів сервісного і сторінкового шару, а інші тести залишаються без змін і продовжують успішно працювати;

– сприяє поділу відповідальност між елементами фреймворку Sitetest та забезпечує максимальне перевикористання коду.

4. Технологія оптимізації процесів тестування веб-сайтів з використанням фрєймворку Sitetest та засобів Sitecore

Технологія оптимізації процесів тестування з використанням фреймворку Sitetest та базових Sitecore компонентів передбачає можливість запуску тестів на різних тестових оточеннях: локально та на сервері. Ініціалізація веб-драйвера здійснюється для цих двох випадків за стандартними алгоритмами, а на виході клієнти класа будуть працювати із IWebDriver об'єктом. Абстрактна стратегія при цьому задається класом WebDriverProvider, що використовує лише один абстрактний метод InitWebDriver(). Конкретними статегіями будуть RemoteWebDriverProvieder та LocalWebDriverProvider, що розширяють клас WebDriverProvider. Статичний клас WebDriverResolver формує та зберігає словник IDictinoary<string, WebDriverProvider>, ключом до якого є значення змінної оточення, що буде надходити до додатку із конфігураційного файлу для двох різних реалізацій базової стратегії. Таким чином, якщо буде потреба додати ще одну логіку ініціалізації драйвера, це можна буде зробити за два простих кроки: створити ще одну імплементацію класу WebDriverProvider, а потім додати створений клас до словника, що зберігається у класі WebDriverResolver.

Тестування за допомогою Sitetest передбачає необхідність створення елементів, їх публікації в базі контент-менеджменті системи і перевірки відображення на веб сайті. Для створення, оновлення, зміни та видалення елементів система тестування використовує API Sitecore Для взаємодії з даними через API реалізуються CRUD операції. CRUD – акронім, що позначає чотири базові функції для роботи з базами даних: створення (create), читання (read), модифікація (update) та видалення (delete).

Всі CRUD операції з компонентами Sitecore виконуються з застосуванням нестандартного класу SitecoreClient за методами классу RestSharp.

SitecoreClient – це нестандартний клас, який може виконувати всі CRUD операції з компонентами Sitecore та використовує методи классу RestSharp. У класі SitecoreClient є набір полів (Language, AuhtorizationCookie і Database), які визначають параметри запиту на Sitecore. Параметр Database параметр передається динамічно під час прогону тестів, а параметр Language прописаний в файлі конфігурації як константа і не налаштовується при запуску автотестів (наприклад: public const string AuhtorizationCookieID = "SessionId"; public const string DatabaseParameterName = "database"; public const string LanguageParameterName = "language").

Логін і пароль адміна для Sitecore зберігаються як поле SitecoreClient. AuthorizationCookie поле задається після виконання логіну на Sitecore. Якщо у cookie файлу закінчується термін дії, то формуються 401 або 403 статус-коди для нашого запиту. Ці статус коди обробляються всередині кожного методу CRUD операцій. Якщо повертається один з цих статус-кодів, то згідно з методом Login() оновлюється значення cookie файлу в поле AuthorizationCookie, а потім вже виконується запит.

Методи GetItemById і GetItemByPath – дозволяють повернення даних про компоненти Sitecore через Id або Path здійснюється за методами GetItemById і GetItemByPath відповідно.

Оскільки для роботи з Sitecore використовується класс RestSharp, то вхідні і вихідні дані при запиті на Sitecore представлені в форматі .json. Застосування різних шаблонів при створення компонентів Sitecore викликає необхідність для кожного з макетів створення шаблону (C # клас), де вказуються імена полів.

Для серіалізації і десеріалізації відповідей і запитів на Sitecore сервері, використовується бібліотека Newtonsoft.Json, що надає гнучкі можливості роботи з .json рядками. Ці рядки можна перетворити в JObject, а потім в екземпляр об'єкта, представленого відповідним шаблоном.

Як приклад, розглянемо форму реєстрації, яка налаштована на веб-сайті із наступним набором полів: EmailLabel, EmailPlaceholder, PasswordLabel, PasswordPlaceholder.

Створюємо класс RegistrationFormSitecoreItem, який буде розширювати класс SitecoreItem (для цього додається атрибут [ItemId ("1234567890")], в якому зазначено id веб-сайту). Для перевірки відображення створених компонентів на сайті будемо використовується модель WebComponentModel, що задає поле моделей компонентів класу.

Для того, щоб порівняти між собою очікуваний результат (контент, налаштований на Sitecore) та актуальний результат (контент на веб-сайті), потрібно створити загальний тип для обох класів: RegistrationFormWebDto. Прошарок автоматизованих тестів DTO (Data Transfer Object), де буде зберігатися модель, є спільний для елементів, набраних

із контент-менеджмент системи та із веб-фрагменту. Клієнт отримує дані, які безпосередньо відображаються у таблицях бази даних. Прошарок Data Transfer Object дозволяє оптимізувати тести таким чином: видалити кругові посилання; приховати певні властивості, які клієнти не повинні переглядати; опустити деякі властивості, щоб зменшити розмір корисного навантаження; вирівняти графіки об'єктів, що містять вкладені об'єкти, щоб зробити їх більш зручними для клієнтів; уникнути вразливостей, пов'язаних із надмірним розміщенням повідомлень; від'єднати рівень обслуговування від рівня бази даних.

Для того щоб уникнути дублювання коду в автотестах використовуються шаблони проектування Page Object та Page Elements.

Page Object – це шаблон проектування, який дозволяє розділяти логіку виконання тестів від їх реалізації. Page Object моделює сторінки тестованого фрагменту програми як об'єкти в коді, що дозволяє визначати окремі класи, які відповідають за роботу з HTM, для кожної конкретної веб-сторінки. Такий підхід значно зменшує обсяг повторюваного коду, тому що одні й ті ж об'єкти сторінок можна використовувати в різних тестах. Основна перевага Page Object полягає в тому, що в разі зміни призначеного для користувача інтерфейсу можна виконати виправлення тільки в одному місці, а не виправляти кожен тест, в якому цей інтерфейс використовується.

Шаблон Page Elements дозволяє ділити сторінки на більш дрібні складові (блоки, віджети тощо), які надалі можна перевикористати в декількох сторінках.

Для візуалізації результату роботи тестів в пропонованій системі тестування використовується фреймворк Allure. Allure Framework – це гнучкий багатомовний інструмент звіту про тести, який не лише формує стисле представлення результатів тестування у формі веб-звіту, але дозволяє тестувальникам отримати максимум корисної інформації з виконання тестів.

Життєвий цикл Allure налаштовується за допомогою файлу json (з ім'ям за замовчуванням allureConfig. json). Пакет NuGet встановлює allureConfig.Template. json, що можна використовувати як шаблон.

Процес тестування можна здійснювати різними способами. Передумовою ефективної автоматизації процесу тестування є тестування в рамках безперервного постачання ПЗ веб-застосунків.

В пропонованій системі тестування може бути використана концепція безперервної інтеграції та доставки (CI/CD), що реалізується як конвеєр, полегшуючи злиття тільки що закоміченного коду в основну кодову базу. Концепція CI/CD дозволяє запускати різні типи тестів на кожному етапі тестування і завершувати його запуском з розгортанням тестованого коду в кінцевий продукт, що отримують кінцеві користувачі (виконання доставки) [16].

На виділеному сервері організовується служба, до завдань якої входять: отримання коду з репозиторію; складання проекту; виконання тестів; розгортання готового проекту у тестовому середовищі; генерація та відправка звітів.

Для налаштування CI/CD обрано інструмент Jenkins. Jenkins надає можливість тестування коду в режимі реального часу та можливість отримувати звіти про окремі зміни в розгорутій кодовії базі. Цей інструмент, головним чином, дозволяє Тестувальникам швидко позначати і виправляти помилки і баги в коді, а потім автоматично тестувати збірку коду.

На рис. 4 наведено узагальнену схему реалізації CI/CD процесу.

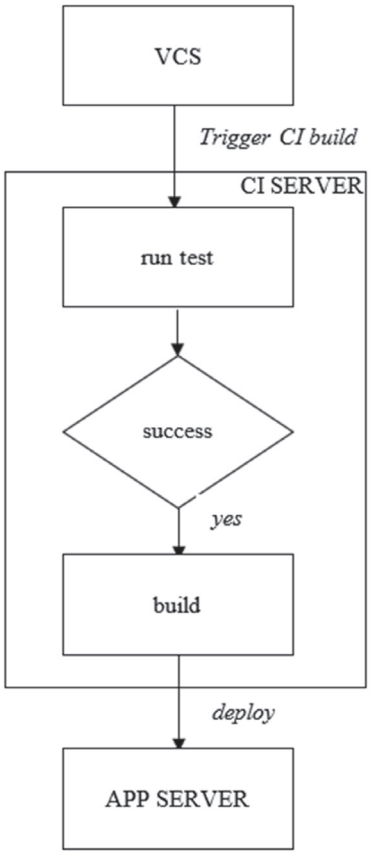

Рис. 4. Узагальнена схема реалізації CI/CD процесу

На рис. 4 використано такі позначення: VCS (англ. Version Control Systems) – система контролю версій; CI server – сервер безперервної інтеграції; APP server (application server) – сервер застосунків системи тестування.

Системи контролю версій VCS значно спрощують роботу з файлами системи тестування та забезпечують відстеження усіх змін. За допомогою VCS можна уникнути втрати даних або виправити помилки, що були допущені в коді або документації.

5. Тестування запропонованої технології

Тестування програмного забезпечення є критичним етапом життєвого циклу розробки ІС та її модулів, що дозволяє виявити недоліки в логіці,

перевірити відповідність функціоналу очікуваним вимогам, а також оцінити стабільність роботи системи за різних умов.

Працездатність запропонованої технології автоматизованого тестування модулів ІС була досліджена на прикладі тестування веб-застосунків (зокрема, інтернет-магазинів). Було розроблено серію тестових сценаріїв, які покривали ключові компоненти клієнтської та серверної частин системи естування. Тестування охоплювало як окремі елементи інтерфейсу, так і повноцінні ланцюги дій, починаючи з авторизації користувача та завершуючи генерацією звітів і аналізом виявлених помилок.

Якість результатів роботи запропонованої технології оцінювалась на основі стандартних метрик в галузі розробки веб-застосунків, з урахуванням особливостей функціонування тестованих модулів. Основна увага приділялася оцінці функціональної коректності, стабільності роботи, швидкодії, безпеки та зручності використання.

Було здійснено порівняння результатів ручного тестування та тестування веб-додатків типу «інтернет-магазин» (https://tochkazp.com.ua), наведених в роботі [14], з результатами тестування веб-сайтів з використанням запропонованого тестового фрєймворку Sitetest, засобів Sitecore CMS та генерації тестових сценаріїв. Визначено, що використання запропонованої технології дозволяє скоротити витрати часу на тестування веб-додатку в середньому в 10 разів в порівнянні з ручним тестуванням та приблизно на 20% в порівнянні з комбінованим використанням генерації тестових сценаріїв і засобів фреймворку Selenium.

Були також змодельовані випадкові спроби завантажити дані для тестування без авторизації (у цих випадках система повертала статус 403 із повідомленням про заборону доступу).

Застосована стратегія тестування дозволила виявити та усунути ряд недоліків на ранніх етапах розробки, а також сформувати уявлення про реальну стабільність і надійність функціонування тестованих модулів.

Висновки

В статті наведено результати розроблення та дослідження технології автоматизованого тестування програмного забезпечення модулів інформаційних систем.

Досліджено різні типи сучасних технології автоматизації тестування, зокрема, тестові фреймворки, що найбільш поширено використовуються для оптимізації процесів тестування веб-застосунків ІС.

До найбільш важливих завдань реалізації запропонованої технології слід віднести: розроблення схеми генерації тестових сценаріїв для автоматичного тестування ПЗ модулів ІС (на прикладі веб-сайтів визначеної предметної області); розроблення технології оптимізації процесів тестування веб-сайтів з використанням запропонованого тестового фреймворку Sitetest та засобів Sitecore.

Наведено схему побудови та використання шаблонів автоматизованого тестування веб-застосунків ІС (для конкретної предметної області).

Розглянуто можливість оптимізації процесів тестування веб-сайтів з використанням спеціалізованого тестового фреймворку та інструментів системи керування контентом Sitecore CMS.

Архітектура запропонованого тестового фреймворку Sitetest містить такі основні шари:

– ядро (Framework Core), що задає конфігурацію веб-драйвера та сукупність конфігураційні класів;

– шар SpecFlow Tests, що шар містить feature-файли та імплементацію кроків для них;

– тестову модель, що містить модуль сторінок, що допомагає інкапсулювати роботу з окремими елементами сторінки, дозволяє зменшити розмір коду та полегшити його підтримку, та сервісний компонент (Page Service), що містить класи, які відповідають за перевірки і взаємодію з елементами сторінок;

– шар Sitecore, що відповідає за взаємодію з Sitecore API та використовується для тестування правильності функціонування додатку зі сторони контент- менеджера.

Наведено докладний опис технології оптимізації процесів тестування веб-чайтів з використанням запропонованого фреймворку.

Працездатність запропонованої технології автоматизованого тестування модулів ІС була досліджена на прикладі тестування веб-застосунків (зокрема, інтернет-магазинів). Визначено, що використання цієї технології дозволяє скоротити витрати часу на тестування веб-додатку в середньому в 10 разів в порівнянні з ручним тестуванням та приблизно на 20% в порівнянні з комбінованим використанням генерації тестових сценаріїв і засобів фреймворку Selenium.

Список літератури

[1] Проектування комп'ютерно-інтегрованих систем : монографія / І. Б. Албанський, Н. Я. Возна, П. В. Гуменний, А. Я. Давлетова, О. М. Заставний – Тернопіль: Університетська думка, 2023. – 494 с.

[2] Поняття, структура та різновиди веб-сайтів. Автоматизоване розроблення веб-сайтів [Електронний ресурс]. – Режим доступу: http://www.ndu.edu.ua/liceum/web.pdf

[3] Monier, M., & El-mahdy, M. M. (2015). Evaluation of automated web testing tools. International Journal of Computer Applications Technology and Research, 4.5, 405–408.

[4] Sampath, S., & Sprenkle, S. (2016). Advances in web application testing, 2010-2014. Advances in Computers, 101, 155–191.

[5] Garousi, V., Mesbah, A., Betin-Can, A., & Mirshokraie, S. (2013). A systematic mapping study of web application testing. Information and Software Technology, 55.8, 1374–1396.

[6] Top 10 Automation Testing Tools in 2021 [Електронний ресурс] – Режим доступу до ресурсу: https://www.netsolutions.com/insights/top-10automation-testing-tools

[7] Модульне тестування [Електронний ресурс]. – 2022– Режим доступу: 208290/mod_resource/content/E2%84%964.pdf.

[8] Єгорова О., Бичок В. Програмні засоби для тестування програмного забезпечення. Молодий вчений. 2019. № 11(75). С. 680–684.

[9] Automation of software testing [Електронний ресурс] – Режим доступу:https://www.sciencedirect.com/science/article/pii/S18770509.

[10] Satheesh, A., & Singh, M. (2017) Comparative study of open source automated web testing tools: Selenium and Sahi. Indian Journal of Science and Technology, 10(13), 1-9.

[11] Kumar, Y. (2015). Comparative study of automated testing tools: Selenium, SoapUI, HP Unified Functional Testing and Test Complete. Journal of Emerging Technologies and Innovative Research, 2.9, 42–48

[12] Sitecore content authoring. [Електронний ресурс]. – 2019 – Режим доступу: https://doc.sitecore.com/users/92/sitecore-experience-platform/en/content-authoring.html/

[13] Test Deliverables in Software Testing – Detailed Explanation [Electronic resource] - Resource access mode: https://www.softwaretestingmaterial.com/test-deliverables/

[14] Удовенко С.Г. Використання шаблонів автоматичного тестування в проектах з розробки веб-додатків / С.Г. Удовенко, Н.О. Міронова, Т.Д. Федорончак, К.К. Верещак // Системи управління, навігації та зв'язку. – 2017. – Вип. 5(45) – С. 111 – 118.

[15] Тер-Карапетянц Т.С. Автоматизація процесу тестування веб-сайтів на платформі Sitecore / Т.С. Тер-Карапетянц, С.Г. Удовенко // Матеріали Міжнародної науково-практичної конференції "Інформаційні технології та системи": тези доповідей, 10-11 квітня 2020 Р. – Х.:ХНЕУ ім. С. Кузнеця, 2020. - С. 29

[16] Rafique, N., Rashid, N., Awan, S., & Nayyar, Z. (2014). Model based testing in web applications. International Journal of Scientific Engineering and Research, 2.1, 56–60.

Надійшла до редколегії 15.04.2025