

Гладун Володимир Романович
кандидат фізико-математичних наук, доцент,
Національний університет “Львівська політехніка”,
Львівський університет бізнесу та права,
<https://orcid.org/0000-0002-4337-8869>

Бешлей Юлія Іванівна
магістр прикладної математики
Національний університет “Львівська політехніка”
<https://orcid.org/0000-0002-9444-3229>

**Створення фреймворку тестування інформаційної системи
оцінювання роботи науково-педагогічних працівників
та визначення їх рейтингів**

JEL Classification: L 860

SECTION “ECONOMICS”: Економіка підприємства та інформаційні технології

Анотація. Сучасне програмне забезпечення – складний багатофункціональний об’єкт, а його якість забезпечується тестуванням на кожному етапі життєвого циклу. Автоматизоване тестування інформаційних систем є сучасною практикою перевірки програмного забезпечення, що включає проведення таких основних дій та кроків тесту, як запуск, ініціалізація, виконання, аналіз і видача результату, автоматично за допомогою спеціалізованих інструментів, з врахуванням технологічних особливостей програмного продукту.

У інформаційній системі оцінювання роботи науково-педагогічних працівників та визначення їх рейтингів виявлено потребу детального тестування з метою покращення якості програмного продукту. Розглянуто планування процесу тестування, яке націлено на те, щоб покриття системи тестами було максимально великим. При цьому, процес перевірки відбувається з точки зору кінцевого користувача, з метою валідації користувацького інтерфейсу. Для цього побудовано UML діаграму діяльності системи та

розглянуто алгоритм генерації тестових сценаріїв на основі стратегії чорної скриньки.

Також розглянуто питання створення фреймворку для автоматизації тестових сценаріїв за шаблоном Page object, опису класів сторінок мовою програмування Java та прив'язка об'єктів до web-елементів через інтегроване середовище Selenium, автоматизації тестів за допомогою бібліотеки TestNG, створення плану запуску автоматичних тестів та збірки проекту через інструмент Gradle.

Ключові слова: методи тестування програмного забезпечення, стратегія чорної скриньки, сценарій тестування, автоматизоване тестування.

Annotation.

Today, humanity has invented an infinite number of things that somehow simplify certain aspects of life, and this applies not only to information technology, but also to industry and everyday life in general. In this age of rapid development of information technology, computers, the Internet, and mobile devices have become an integral part of our lives. However, the basic purpose of these things has not changed, almost from the very beginning of their creation - they are intended to facilitate the manual and mental work of man. In addition, the computer system is capable of eliminating the human factor and of making calculations without fail, provided that it is properly designed.

Given the high level of competition in the IT product market, customers require quality software. To meet consumer needs, software developers hire professional quality engineers and analysts whose main task is to control the quality of the product.

Modern software is a complex multifunctional object, and its quality is ensured by testing at every stage of the life cycle. Automated testing of information systems is a modern practice of software validation, which involves performing such basic actions and steps of the test like launching, initializing, executing, analyzing and delivering the result, using specialized tools automatically, considering technological features of the software product.

In the information system for evaluating the work of scientific and pedagogical staff and determining their ratings revealed the need for detailed testing for the purpose to improve the quality of the software product. Reviewed planning of the testing process is considered, that aims to maximize the coverage of the system by tests. Herewith, the validation process takes place from the end-user perspective, in order to validate the user interface. For this purpose, a UML diagram of the system activity was constructed and an algorithm for generating test scenarios based on a black box strategy was considered.

Also discussed the problem of creation of a framework for automation of Page object test scripts, description of Java class pages, and binding of objects to web elements through the Selenium integrated environment, test automation using the TestNG library, creation of an automatic test run and assembly plan project through the Gradle tool.

Key words: software testing methodologies, black-box strategy, testing scenario, test automation.

Вступ

На сьогодні людство винайшло незмірну кількість речей, що так чи інакше спрощують певні аспекти життєдіяльності, причому це стосується не тільки інформаційних технологій, а й промисловості та повсякденного життя в цілому. В епоху доволі стрімкого розвитку інформаційних технологій комп'ютери, інтернет, мобільні пристрої стали невід'ємною частиною нашого життя. Проте основне призначення цих речей не змінилося, ледь не з самого початку їх створення – вони покликані полегшувати ручну та розумову працю людини. Окрім того, комп'ютерна система здатна виключати людський фактор та безпомилково здійснювати обчислення, за умови коректності її розробки.

За умов високої конкуренції на ринку ІТ продуктів, замовники вимагають якісного програмного забезпечення (ПЗ). З метою задоволення потреб споживача розробники ПЗ наймають професійних інженерів з якості та аналітиків, основним завданням яких є контроль якості продукту.

Однією з технік контролю якості ПЗ є тестування [1]. Процес тестування складається з етапів планування робіт (Test Management), проектування тестів (Test Design), виконання тестування (Test Execution) та аналізу отриманих результатів (Test Analysis). Тестування дозволяє перевірити, чи правильно реалізовано усі вимоги до ПЗ та допомагає у виявленні помилок, забезпечує їх розпізнавання до етапу розгортання.

Результати дослідження

Автоматизоване та мануальне тестування. Тестування ПЗ має два основних напрямки: автоматизоване та мануальне (ручне). Правильне планування тестування, допомагає заощадити часові та грошові ресурси розробника ПЗ [2]. Обидві методики тестування мають свої переваги і недоліки.

Мануальне тестування може потребувати багато часу, але в короткостроковій перспективі заощадить більше коштів. Вартість такого тестування залежить лише від інженера з якості (QA Engineer) і не залежить від інструментів для автоматизації. Мануальне тестування можна розглядати як взаємодію інженера з якості і безпосереднього розробника ПЗ з метою пошуку дефектів та помилок. Взаємодіючи з додатком безпосередньо, інженер з якості може порівнювати очікуваний результат з реальним і залишати рекомендації для розробника ПЗ. До основних переваг ручного тестування належать: 1) дешевизна – у короткостроковій перспективі ручне тестування дешевше, ніж інструменти автоматизованої перевірки, 2) тестування в реальному часі – незначні зміни можуть бути досліджені відразу, без написання коду і його виконання, 3) можливість дослідницького тестування – його метою є перевірка різноманітних можливостей додатка, важливо, що використовуються не заздалегідь складені тест-кейси, а придумані на льоту сценарії. Недоліками ручного тестування є: 1) людський фактор – люди інколи схильні до

неефективності, відповідно деякі помилки можуть залишитися невиявленими, 2) трудомісткість повторного використання – виконати серію стандартних автоматичних тестів простіше, ніж протестувати проект вручну після внесення навіть невеликих змін, 3) неможливість тестування навантаження – відсутність можливості вручну змодельовати велику кількість користувачів.

Автоматизоване тестування – використовує програмні засоби для виконання тестів і перевірки результатів виконання, що допомагає скоротити час тестування і спростити його процес. Із його допомогою очікувані сценарії порівнюються з тим, що отримує користувач, вказуються відмінності. Автоматизоване тестування відіграє важливу роль у великих проектах з великою кількістю функцій. До переваг автоматизованого тестування належить: 1) можливість тестування навантаження на інформаційну систему (ІС), 2) заощадження часу, 3) можливість повторного використання – тестовий сценарій, написаний один раз, може бути використаний в майбутньому при черговому оновленні проекту. Недоліками автоматизованого тестування є: 1) висока вартість – інструменти автоматизованого тестування, а також навчання стосовно їх використання коштують відносно дорого, тому потрібно ретельно оцінювати бюджет проекту, 2) автоматизоване тестування не може повністю покрити вимоги до інтерфейсу користувача, 3) відсутність «людського погляду» – у ПЗ можуть існувати помилки, які виявить лише людина [3].

Стратегії чорної та білої скриньок. Стратегії чорної та білої скриньок належать до динамічних технік тестування ПЗ. Тестування з використанням стратегії білої скриньки (white-box testing) або структурне тестування полягає у детальному дослідженні вихідного коду програми – внутрішньої логіки і структури програми. При використанні стратегії білої скриньки розробник тесту має доступ до вихідного коду програм і може писати код, який пов'язаний з бібліотеками ПЗ, що тестується. Цей підхід використовують при юніт-тестуванні (unit testing), при якому тестуються лише окремі частини ІС. Існує три методи тестування за стратегією білої скриньки: тестування на основі потоку керування програми, на основі потоку даних та мутаційне тестування.

Тестування з використанням стратегії чорної скриньки (black-box testing) або функціональне тестування не вимагає знань про внутрішню роботу ПЗ. При тестуванні за цією стратегією, інженер має доступ до ПЗ лише через ті ж інтерфейси, що і замовник або користувач, або через зовнішні інтерфейси, що дозволяють іншому комп'ютеру або іншому процесу підключитися до системи для тестування. Найпоширенішими методами функціонального тестування є методи випадкового тестування, еквівалентної розбивки й аналізу граничних умов. Метод еквівалентної розбивки полягає у розбитті множини значень вхідних даних на скінченну кількість класів еквівалентності так, щоб кожний тест, що є представником певного класу, був еквівалентним будь-якому іншому тесту цього класу. Два тести вважають еквівалентними, якщо вони виявляють однакові помилки. Проектування автоматизованих тестів за цим методом має два етапи: побудова класів еквівалентності та побудова множини тестів. Цей

метод тестування дозволяє зменшити кількість тестів у порівнянні з методом випадкового тестування [4].

Вибір стратегії білої скриньки чи стратегії чорної скриньки залежить від того, чи має розробник тестів доступ до вихідного коду ПЗ, або ж тестування виконується через інтерфейс користувача або прикладний програмний інтерфейс. Існує також метод сірої скриньки, що поєднує в собі обидва попередніх підходи [5].

У роботі розглядається питання побудови та автоматизації множини тестів користувацького інтерфейсу інформаційної системи «Оцінювання роботи науково-педагогічних працівників кафедри та визначення їх рейтингів», при цьому використовується стратегія чорної скриньки, тобто до уваги не береться внутрішня логіка продукту. Завданням цієї системи є спрощення і автоматизація процесу фіксування та оцінювання результатів професійної діяльності науково-педагогічних працівників навчального закладу вищої освіти. Цей веб-додаток є інформаційною системою, з можливим великим навантаженням, для користувачів, що можуть у ній авторизуватись та мати певні користувацькі права. Усі тестові процеси у цій системі автоматизовані – це достатньо зручно для швидкої ідентифікації дефектів та заощадить часові ресурси, в порівнянні з мануальним тестуванням.

У роботі [6] запропоновано метод автоматизованої побудови сценаріїв тестування ПЗ стратегіями чорної та білої скриньки на основі моделі поведінки програмного продукту, що базується на аналізі його змінних. При побудові тестових сценаріїв за сценарієм чорної скриньки, вважають, що кожен сценарій тестування T^k складається з декількох кроків $T^k = \{T_0^k, T_1^k, \dots, T_n^k\}$, де k – номер тесту, крок сценарію $T_j^k = (S_j^k, C_j^k)$, $j = \overline{0, n}$ – це пара методу S_j^k та множини змінних C_j^k з відповідними класами еквівалентності. На кожному кроці тесту T_j^k змінні набувають значень внаслідок введення даних користувачем ПЗ, що може призвести до помилок під час виконання сценарію тестування.

Розглянемо цей метод на прикладі досліджуваної ІС. На рис. 1 зображено UML-діаграму діяльності для ІС. Діаграму діяльності можна ефективно використовувати при проектуванні тестових сценаріїв, оскільки вона демонструє методи – дії, які може зробити користувач. Ця діаграма є дещо спрощеною, наприклад, на ній відсутні можливості скасування операцій. При створенні фреймворку тестування враховано цю властивість системи.

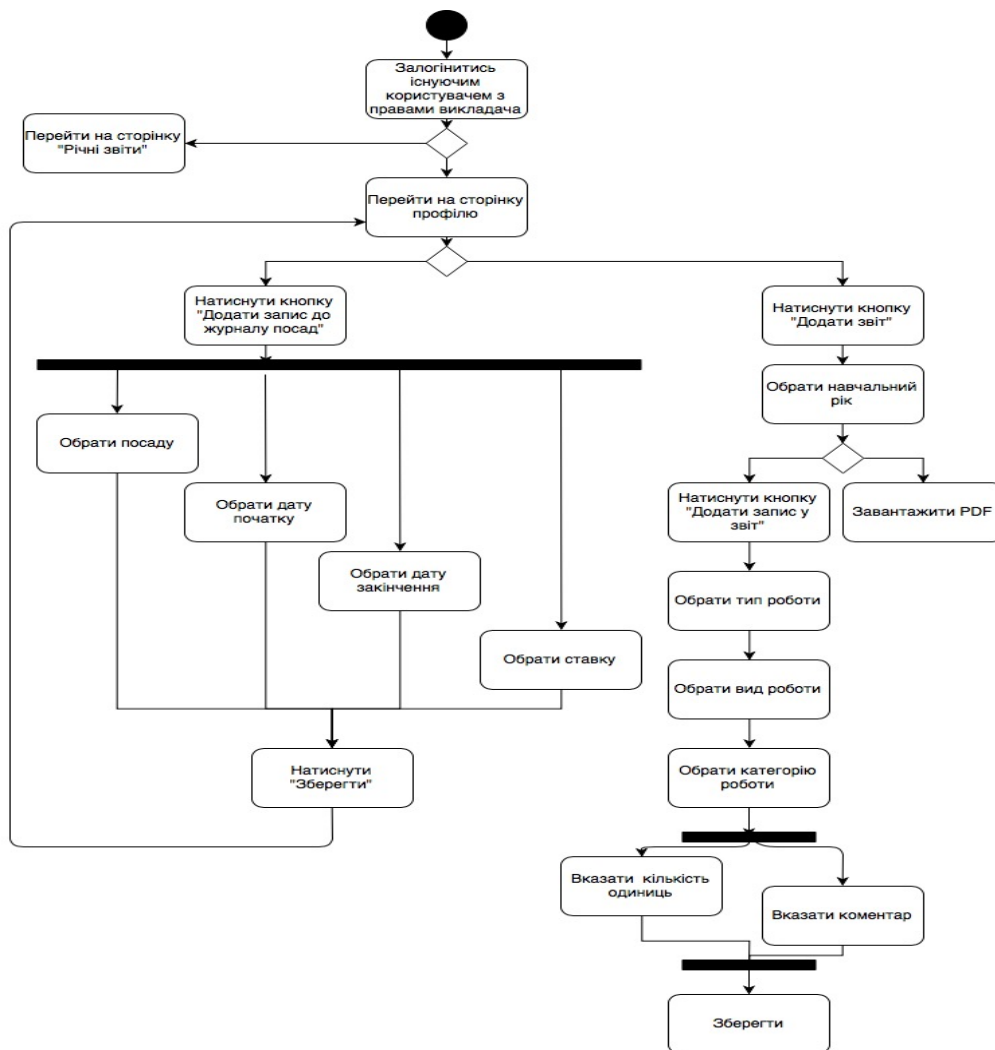


Рис. 1. Діаграма діяльності інформаційної системи.

Нехай TS – набір усіх сценаріїв тестування. На початку $TS = \{\}$. Побудуємо перший тестовий сценарій додавання користувачем запису до журналу посад:

$T_0^0 = (\text{"Залогуватись з правами викладача"}, \text{"test@gmail.com:testrate"})$

$T_0^1 = (\text{"Перейти на сторінку профілю"}, \text{null}).$

$T_0^2 = (\text{"Натиснути кнопку додати запис до журналу посад"}, \text{null});$

$T_0^3 = (\text{"Обрати посаду"}, \text{доцент});$

$T_0^4 = (\text{"Обрати дату початку"}, 01.07.2018);$

$T_0^5 = (\text{"Обрати дату завершення"}, 30.06.2019);$

$T_0^6 = (\text{"Вказати ставку"}, 1.00);$

$T_0^7 = (\text{"Натиснути кнопку зберегти"}, \text{null}).$

За таким принципом побудовано повний набір сценаріїв тестування.

Наступний етап – автоматизація тестування. Розглянемо інструменти та технології, використані при проектуванні та реалізації фреймворку тестування ІС.

Тестування з використанням стратегії чорної скриньки автоматизовано за допомогою драйвера браузера Selenium WebDriver. Selenium – це інструмент для автоматизованого керування настільними та мобільними браузерами.

Selenium підтримується виробниками усі популярних браузерів. Вони адаптують браузери для інтеграції з Selenium, а іноді реалізують вбудовану підтримку цього інструмента у браузері. Компонентом інструмента Selenium є Selenium WebDriver, що містить набір клієнтських бібліотек на різних мовах програмування, зокрема мові Java, і дозволяє керувати браузером на програмному рівні [7].

TestNG – інструмент для запуску тестів, який працює за принципом анотацій і використовується разом з пакетом Selenium. Анотація в TestNG – це мітка у об'єктно-орієнтованій мові програмування, яка не виконує жодних дій і використовується як вказівник. За допомогою анотацій, тести можна групувати різними способами або надавати їм різні пріоритети, наприклад один тест може бути у кількох групах тестів. Тому TestNG дає можливість декларувати різноманітні передумови до виконання тестів для різних груп. TestNG забезпечує гнучкість виконання тестів на різних наборах даних, через файл TestNG.xml або через концепцію постачальників даних (data-provider).

Ще однією важливою можливістю цього інструменту є генерування звітів про результати виконання тесту (успішне, некоректне чи неуспішне виконання тесту). Успішно виконаним тестом вважається коректний тест, при виконанні якого система пройшла перевірку. Виконаним некоректно тестом вважається тест, при виконанні якого виникла помилка, яка не стосується системи що тестується. Неуспішним тестом є тест, який система не пройшла, внаслідок наявної помилки у ній.

Page Object – одне з найбільш використовуваних архітектурних рішень в автоматизації. Цей шаблон проектування допомагає інкапсулювати роботу з окремими елементами сторінки, що дає можливість зменшити кількість коду та полегшити його підтримку. Якщо дизайн однієї зі сторінок був змінений, то потрібно буде лише змінити відповідний клас, що описує цю сторінку. Таким чином, тести будуть працювати не з низькорівневим кодом, а з високорівневою абстракцією.

Для того, щоб досягти до елементів веб-сторінки, необхідно мати доступ до браузера. Таку можливість дає драйвер, який пов'язує мову програмування з елементами веб-сторінки у браузері. Для кожного браузера використовується свій драйвер. Унікальний шлях, який вказує драйверу місцезнаходження веб-елемента називається локатором. Отже, головна ідея шаблону полягає у створенні класу на певній мові програмування, полями якого будуть локатори, а методами – функціональні дії зі сторінкою, що визначаються кроками у тестовому сценарії.

Розглянемо переваги використання даного шаблону. Шаблон забезпечує розділення вповноважень: вся логіка сторінки зосереджена в Page Object класах, а тестові класи використовують лише їх відкриті методи та перевіряють результат. Також, вагомою перевагою є те, що усі локатори розміщені в класі сторінки. Наступна перевага – інкапсуляція роботи з драйвером, яка дає можливість реалізувати крос-платформну автоматизацію тестів.

В класичному варіанті, створюють один клас на одну веб-сторінку. Але не завжди такий підхід є зручним. Сторінка може бути складною і містити

багато динамічних веб-елементів. Або ж на усіх сторінках може бути присутнім один статичний елемент, як, наприклад, панель навігації у досліджуваній системі (рис. 2). У цьому випадку важливо налаштувати шаблон під особливості системи, що тестується. Наприклад, для того, щоб метод, який буде натискати на кнопку “Викладачі” був доступним для усіх сторінок фреймворку автоматизації, достатньо створити окремий клас, який буде вершиною ієрархії сторінок. У фреймворку тестування це реалізовано за допомогою принципу наслідування у об’єктно-орієнтованій мові програмування Java.

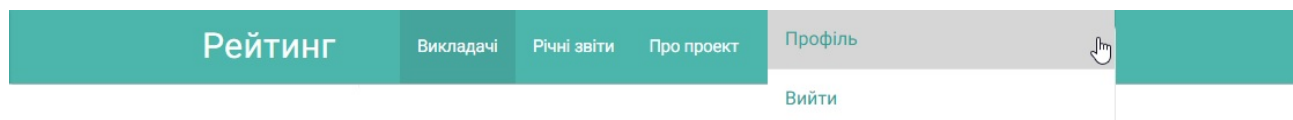


Рис. 2. Панель навігації.

Архітектурний шаблон містить також поняття n -рівневої структури тестового фреймворку. На найнижчому рівні спроектовано класи сторінок, які містять опис веб-сторінки через локатори та атомарні функції. На наступному рівні спроектовано класи, які реалізують бізнес логіку кроків тестового сценарію. Ці методи мають містити виклики функцій з попереднього рівня. На останньому рівні спроектовано тест-класи. Зазначимо, що кількість рівнів може бути довільною і залежить від специфіки програмного забезпечення, що тестується. Найчастіше обирають 2 або 3-рівневу структуру проекту. Також у фреймворку використовуються допоміжні класи, наприклад клас під’єднання до бази даних.

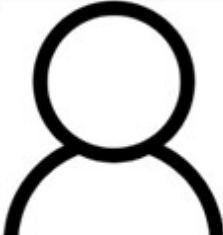
Зауважимо, що відсутність архітектури проектування тестового фреймворку може призвести до неефективного переписування коду, що в свою чергу потребує часових затрат.

Розглянемо опис веб-сторінки у вигляді класу на прикладі сторінки профілю викладача (рис. 3).

Верхню панель меню доцільно описати у окремому класі Page, оскільки вона доступна для всіх веб-сторінок системи. Page – це суперклас для усіх інших сторінок. Тобто з будь-якої сторінки можна досягти до методів верхньої панелі меню.

Викладач Тестовий

Коротка інформація



тестовий викладач, який перевірятиме роботу системи

РЕДАГУВАТИ ПРОФІЛЬ

Журнал посад

[+ ДОДАТИ ЗАПИС](#)

Викладач Викладач Тестовий ще не має записів у журналі посад.

Річні звіти

[+ ДОДАТИ ЗВІТ](#)

Рис. 3. Сторінка профілю викладача.

Клас сторінки профілю викладача `ProfilePage` є нащадком класу `Page` та описує ім'я та прізвище викладача, кнопки редагування профілю, додавання запису в журнал посад, додавання звіту. Описати ці веб-елементи можна за допомогою локаторів.

Важливим також є регулювання пауз. Оскільки тести можуть виконуватись в різних середовищах, то їх успішне виконання залежить від багатьох факторів, один з яких швидкість інтернет-з'єднання. Для цього створено допоміжний клас, який надає програмний інтерфейс для роботи з паузами. Зокрема цей клас надає можливість очікувати поки веб-елемент стане видимим чи клікабельним (clickable).

Для опису користувача системи створено клас `User`. Ця сутність має такі поля: ім'я, користувацьке ім'я, пароль. Для створення об'єктів класу `User` використано шаблон "Репозиторій", що містить методи, що повертають вже проініціалізований об'єкт `User`.

Наступним кроком автоматизації тестового сценарію є створення тест-класу функціональних методів, що забезпечують виконання кроків тестового сценарію. Отже, розглянемо один із тестових сценаріїв:

Прередумови:

1. Відкрити браузер.
2. Перейти за посиланням: <http://lecturers-rating.tk>.
3. Залогуватись з правами викладача.
4. Перейти на сторінку профілю викладача.

Кроки:

1. Натиснути кнопку «Додати запис» до журналу посад.
2. Заповнити посаду значенням «Доцент».
3. Обрати дату початку роботи “01.07.2018”
4. Обрати дату завершення роботи “30.06.2019”.
5. Встановити ставку “1.00”
6. Натиснути кнопку «Зберегти»
7. Перейти на сторінку профілю та перевірити чи вся інформація введена коректно

Післяумови:

1. Видалити запис
2. Закрити браузер

Оскільки багато тестових сценаріїв використовують передумови та післяумови цього сценарію, то їх реалізовано в окремому класі “TestSetup”. Клас “JobLogTestSuite” є нащадком класу “TestSetup” і містить методи, які забезпечують кроки цього тестового сценарію (рис. 4).



Рис. 4. Клас сторінки профілю викладача.

Висновки

У роботі розглянуто процес тестування ПЗ, що є важливою частиною життєвого циклу програмного продукту. Розглянуто також особливості різних методів тестування ПЗ.

Розглянуто питання побудови та автоматизації сценаріїв тестування з використанням стратегії чорної скриньки користувацького інтерфейсу інформаційної системи «Оцінювання роботи науково-педагогічних працівників кафедри та визначення їх рейтингів».

При створенні фреймворку тестування використано сучасні технології імплементації автоматизації тестових сценаріїв. Побудовано архітектуру фреймворку за найвідомішим у автоматизації шаблоном Page Object, мовою програмування Java створено класи сторінок ІС. Створено програмний

інтерфейс, що надає доступ до зберігання та використання тестових даних та написання тестових сценаріїв за заданими кроками. Наприкінці кожного тесту реалізовано перевірки, що дають можливість отримати звіт про кількість успішних тестів.

Список використаних джерел

1. D. Graham, E. Van Veenendaal, I. Evans, R. Black. Foundations of Software Testing: ISTQB Certification. – Cengage Learning Emea, 2008. – 258 p.
2. D. Kumar. The Impacts of Test Automation on Software's Cost, Quality and Time to Market / D. Kumar, K. K. Mishra // Procedia Computer Science. – 2016. – No 79. – 8-15.
3. M. Fewster , D. Graham. Software Test Automation: Effective Use of Test Execution Tools. – Addison Wesley, 1999. – 574 p.
4. Hu Y.T. Automatic Black-Box Method-Level Test Case Generation Based on Constraint Logic Programming. / Y.T. Hu, N.W. Lin // Computer Symposium (ICS). – 2010. – pp. 977-982.
5. M. E. Khan. A Comparative Study of White Box, Black Box and Grey Box Testing Techniques / M. E. Khan, F. Khan // IJACSA. – 2012. – Vol. 3, No.6. – pp. 12-15.
6. Д. В. Федасюк. Метод побудови сценаріїв тестування програмного забезпечення на основі аналізу його змінних / Д. В. Федасюк, В. С. Яковина, П. В. Сердюк, О. О. Нитребич // Інформаційні технології та комп'ютерна інженерія. – 2014. – № 2. – С. 50–58.
7. D. Kovalenko. Selenium Design Patterns and Best Practices. – Packt Publishing Limited. – 2014. – 270 p.