УДК: 681.5 DOI 10.33111/mise.103.8

Деменко І.О.,
магістрант за освітньо-професійною програмою «Системний аналіз»
КНЕУ імені Вадима Гетьмана
Камінський О.Є.,
д. е. н., доцент, доцент кафедри системного аналізу та кібербезпеки
КНЕУ імені Вадима Гетьмана

Demenko I.O,
master's student in the educational and professional program
«System Analysis»
KNEU named after V. Hetman
Kaminsky O.E,
Doctor of Economics, Associate Professor,
Associate Professor of the Department of System Analysis and Cybersecurity,
KNEU named after V. Hetman

АНАЛІЗ ВПЛИВУ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ НА ЯКІСТЬ ТА БЕЗПЕКУ ПЗ

ANALYSIS OF THE IMPACT OF AUTOMATED TESTING ON SOFTWARE QUALITY AND SECURITY

Анотація. *Дослідження впливу автоматизованого тестування програмного забезпечення є важливою та актуальною частиною досліджень у сфері програмної інженерії і розробки. Повсюдне використання та впровадження інформаційних технологій у нашому житті сприяє зростанню уваги до якості програмного забезпечення, тому питання впровадження автоматизованого тестування виходить на передовий план. У контексті швидкоплинного технологічного ландшафту та постійного зростання складності IT-проектів, питання вибору підходу до тестування програмного забезпечення набуває важливості та актуальності. Однією з ключових стратегій є використання автоматизованого тестування, проте визначення його продуктивності та придатності для конкретного проекту виявляється завданням, що вимагає обґрунтованого підходу та уважного аналізу. Метою статті є здійснення аналізу впливу автоматизованого тестування на розробку IT-проекту на реальному прикладі. В роботі наведено реальний приклад впровадження автоматизованого тестування для IT-проекту, проведено ретельний та детальний аналіз його впливу на якість і ефективність розробки IT-проекту. Практичною значущістю є можливість використання створеної роботи як інструкції для IT-проектів, які використовують, або збираються використовувати автоматизоване тестування у майбутньому.*
Ключові слова. *Розробка IT-проектів, програмне забезпечення, тестування, автоматизоване тестування, тести, тестові сценарії, аналіз даних.*

Abstract. *The study of the impact of automated testing of software is an important and relevant part of research in the field of software engineering and*

development. The widespread use and implementation of information technologies in our lives contributes to the growing attention to the quality of software, therefore the issue of the introduction of automated testing comes to the fore. In the context of the fast-moving technological landscape and the constant increase in the complexity of IT projects, the issue of choosing an approach to software testing is gaining importance and relevance. One of the key strategies is the use of automated testing, but determining its performance and suitability for a particular project is a task that requires a sound approach and careful analysis. The purpose of the article is to analyze the impact of automated testing on the development of an IT project using a real example. The work presents a real example of the implementation of automated testing for an IT project, and a thorough and detailed analysis of its impact on the quality and efficiency of IT project development. Of practical significance is the possibility of using the created robot as instructions for IT projects that use or are going to use automated testing in the future.
Keywords: *IT project development, software, testing, automated testing, tests, test scenarios, data analysis.*

Постановка проблеми. У сучасному світі, що стрімко розвивається в галузі інформаційних технологій, розробка програмного забезпечення стає необхідністю для багатьох компаній та організацій. З метою забезпечення високої якості продукції та швидкого випуску на ринок, велика увага приділяється процесам тестування. Однак, разом зі зростанням складності та обсягів інформаційних технологічних проектів, виникає необхідність у вдосконаленні методів тестування. Однією з актуальних тенденцій в сфері тестування програмного забезпечення є використання автоматизованих засобів тестування. Автоматизація тестування дозволяє підвищити ефективність процесу, скоротити терміни розробки, та зменшити ймовірність помилок. Незважаючи на очевидні переваги, що вносить автоматизоване тестування, досі існують ряд невирішених проблем та питань, пов'язаних з його впливом на загальний процес розробки програмного забезпечення.

У цьому контексті, **метою даної статті** є аналіз впливу автоматизованого тестування на розробку програмного забезпечення, зокрема процесу впровадження автоматизованого тестування. Дослідження має на меті зробити вагомий внесок у покращення методів тестування програмного забезпечення та сприяти розвитку більш ефективних практик у сфері IT-розробки.

Аналіз останніх джерел та публікацій. На сьогоднішній день існує велика кількість наукових робіт, які висвітлюють різні підходи до впровадження автоматизованого тестування під час розробки IT-проектів, порівнюють їх між собою, виділяють їхні особливості. Так, наприклад, наукові роботи Роберта К. Мартіна, Джеймса В. Ньюкірка, Роберта С. Косса «Швидка розробка про-

грам. Принципи, приклади, практика» [2] та Shravan Pargaonkar «A Comprehensive Research Analysis of Software Development Life Cycle (SDLC) Agile & Waterfall Model Advantages, Disadvantages, and Application Suitability in Software Quality Engineering [1] присвячені прискіпливому вивченню найбільш популярних та ефективних підходів по впровадженню автоматизованого тестування в цикли розробки IT-проектів.

З одного боку, автоматизація може значно зменшити час, необхідний для виконання тестування, забезпечуючи швидше виявлення та виправлення помилок. З іншого боку, автоматизоване тестування може вимагати значних витрат на впровадження та обслуговування, що може бути нецільовим для деяких проектів. Дані проблеми дуже детально розглядається та аналізується у таких роботах як «Повний посібник з автоматизації тестування» Аксельрод А. [3] та «Аналіз сучасних методологій розробки складних програмних проектів» Говорущенко Т. О. [4].

У роботі [17] представлено детальне обговорення завдань тестування програмного забезпечення, для яких зазвичай використовуються LLM (великі мовні моделі), проаналізовано широко використовувані LLM, види оперативного проектування, які використовуються, а також супутні методи, які застосовуються поряд з штучним інтелектом.

Даний напрямок досліджень є актуальним і велика кількість науковців проявляє до нього зацікавленість. Але, на превеликий жаль, у сучасному світі все дуже швидко змінюється та розвивається, тому не можливо дати рішення для всіх проблем, і залишається простір для подальших наукових досліджень.

Виклад основного матеріалу дослідження. Почнемо з визначення терміну «тестування». Тестування — це процес перевірки відповідності функціональності продукту заявленим вимогам, що здійснюється через спостереження за його роботою в умовах, які створюються штучно, а також за допомогою обмеженого набору тестів, обраних специфічним чином.

Автоматизоване тестування, часто називане автоматизацією тестування, є методом випробування програмного забезпечення, який використовує спеціальні програмні засоби для виконання заданого набору тестових сценаріїв. Програмне забезпечення для автоматизації тестування може також вводити тестові дані в тестове середовище, порівнювати очікувані та фактичні результати, а головне — генерувати докладні звіти про тести. Однак важливо зауважити, що автоматизація тестування може вимагати значних інвестицій у грошовому виразі та ресурсах.

У великих компаніях, таких як Google, Facebook, Microsoft та інших, де процес розробки проходить через послідовні цикли, використання автоматизованого тестування стає невід'ємною частиною розробки. Це дозволяє записати та відтворити набір тестів необмежену кількість разів за необхідності, що спрощує процес тестування в рамках повторюваних циклів розробки.

Найголовніше процес автоматизованого тестування пришвидшує процес розробки IT-проекту, за рахунок швидкого виявлення дефектів у роботі з меншими ризиками людських помилок.

Розглянемо, як приклад, IT-проект, який є складовою діяльності корпорації — світового лідера у сфері інформації, аналітики та рішень для основних галузей і ринків, що сприяють розвитку економіки по всьому світу. Компанія співпрацює з клієнтами у сфері бізнесу, фінансів та урядової діяльності, щоб допомогти їм побачити загальну картину з неперевершеними знаннями, які ведуть до обґрунтованих і впевнених рішень. Корпорація обслуговує понад 30000 ключових клієнтів у більш ніж 100 країнах світу. Головна ціль такого проекту — це розробка та тестування програмного забезпечення, а саме розробка інформаційної підсистеми для банків. IT-проект розбитий на складові частини, відповідно існує велика кількість команд, кожна відповідає за конкретний компонент величезної системи та займається його тестування та розробкою.

Із самого початку проекту був період, коли робота виконувалася повільно та непродуктивно, розробка виконувалася із затримкою, тестування було неякісним. Це все негативно впливало на замовника та робітників, становище в команді напружувалося, потрібні були швидкі, кардинальні, а головне ефективні зміни. Саме тоді було прийнято рішення впровадити автоматизоване тестування в процес розробки IT-проекту. Менеджмент прийняв правильне рішення і це допомогло подолати складне становище на проекті та вивело його на новий якісний рівень. Розглянемо перелік дій, які допомогли ефективно впровадити автоматизоване тестування для IT-проекту:

1.Перед початком впровадження автоматизованого тестування дуже важливо ретельно проаналізувати вимого та розробити якісну тестову документацію, адже це буде фундаментом для процесу тестування. Аналіз вимог є важливим етапом у процесі підготовки до автоматизованого тестування IT-проекту. Цей етап дозволяє ретельно вивчити функціональні та нефункціональні вимоги до системи та визначити стратегію тестування, щоб забезпечити повноту та якість покриття тестами. Під час аналізу

вимог визначається обсяг тестування, яке включає в себе всі аспекти функціональності та надійності системи. Тестування повинно охоплювати всі можливі сценарії використання системи, а також перевіряти її взаємодію з різними складовими інфраструктури. Також аналіз включає врахування нефункціональних вимог, таких як продуктивність, масштабованість, безпека та інші аспекти. Майбутні сценарії повинні тестувати відповідність системи цим вимогам та перевіряти, чи виконується кожна з них.

2.На основі проведеного аналізу розробляється тестова документація. Тестова документація — це набір документів, які визначають стратегію тестування, план тестування, тест-кейси, сценарії тестування та інші важливі аспекти тестування програмного забезпечення. Основна мета тестової документації полягає в тому, щоб забезпечити повноту тестування та визначити, чи відповідає програмне забезпечення вимогам та очікуванням.

3.Після того як всі вимоги було проаналізовано та розроблено тестову документацію, є можливість приступити до впровадження автоматизованого тестування для нашого проекту. Даний крок є ключовим в розробці програмного забезпечення, спрямованим на підвищення якості, швидкості та ефективності тестового процесу.

Можна виділити наступні етапи впровадження автоматизованого тестування:

- **Аналіз та планування** — відбувається аналіз та вибір тестових інструментів для автоматизованого тестування, розробляється план дій для забезпечення ефективної розробки тестових сценаріїв для покриття функціоналу IT-проекту.
- **Підготовка тестового середовища та налаштування тестових даних** — відбувається підняття тестових середовищ, таких як тестові сервери, бази даних тощо.
- **Розробка тестових сценаріїв** — відбувається розробка та написання автоматизованих тестів, за допомогою обраних інструментів.
- **Виконання та аналіз результатів** — відбувається запуск тестів та аналіз результатів для знаходження дефектів у роботі IT-проекту.
- **Впровадження у виробництво** — інтегрується процес автоматизованого тестування у виробництво для запуску сценаріїв на регулярній основі.
- **Підтримка та оптимізація** — впроваджується моніторинг автоматизованих тестів для їхньої підтримки та оптимізації, щоб забезпечити високий рівень актуальності та ефективності. Також

на даному етапі відбувається виправлення дефектів та збоїв у тестах для забезпечення стабільної роботи.

Кожен з вищезазначених етапів відіграє важливу роль у успішному запуску автоматизованих тестів.

Після впровадження автоматизованого тестування зазвичай починається процес аналізу виконаної роботи. Для початку н потрібно відібрати необхідні дані та створити Dataset. Dataset — це набір даних або колекція інформації, яка групується разом з метою подальшого аналізу чи обробки. Dataset може включати в себе різні типи даних, такі як текст, числа, зображення, аудіо, відео та інше.

Після збору всієї необхідної інформації, даних для аналізу, розробляється Dataset, який містить наступні атрибути:

—Number Of Sprint — під час розробки IT-проекту була використану гнучка методологія Scrum, тому життєвий шлях проекту розбивається на умовні спринти. Спринт — це основна одиниця часу в методології Scrum, яка використовується для планування, розробки та постійного вдосконалення продукту, спринт має фіксовану тривалість, у нашому випадку це два тижні.

—Total Number Of Scenarios — загальна кількість тестів на момент спринту.

—Pass Rate — це відсоток успішно пройдених тестів відносно загальної кількості виконаних тестів. Ця метрика використовується для оцінки якості тестового покриття та ефективності тестування.

—Number Of Failed Scenarios — кількість тестових сценаріїв, що провалилися під час запуску.

—Critical Bugs — це кількість знайдених дефектів з критичним пріоритетом. Критичні помилки визначаються як серйозні дефекти чи помилки в програмному забезпеченні, які мають суттєвий вплив на його функціональність, безпеку чи інші критичні аспекти. Ці помилки можуть призводити до серйозних проблем і негативно впливати на користувачів, бізнес-процеси чи безпеку системи в цілому.

—High Bugs — це кількість знайдених дефектів з високим пріоритетом. Це дефекти чи помилки в програмному забезпеченні, які, хоча і не є критичними для його функціональності чи безпеки, все ж можуть мати суттєвий вплив на роботу системи чи користувачів. Вони визначаються як проблеми, які потребують уваги та виправлення, але не мають такої негайної важливості, як критичні помилки.

—Medium Bugs — це кількість знайдених дефектів з середнім пріоритетом. Це дефекти чи помилки в програмному забезпеченні, які мають помірний вплив на функціональність, безпеку чи використання продукту. Вони не є критичними, але вимагають уваги та виправлення для покращення якості продукту.

—Low Bugs — це кількість знайдених дефектів з низьким пріоритетом. Це дефекти чи помилки в програмному забезпеченні, які мають мінімальний вплив на функціональність, безпеку чи використання продукту. Вони є менш критичними порівняно з вищими рівнями важливості, і їх виправлення може бути відкладене на майбутні версії програми.

—Test Coverage % — це метрика, яка вимірює обсяг коду програми, який був протестований сценаріями. Вона вказує на те, скільки частин програмного коду було викликано тестами під час їх виконання.

Після побудови Dataset, для його аналізу, для цього використаємо такий інструмент як Python. Python –потужна та гнучка мова програмування, яка отримала широке визнання в області аналізу даних.

Для початку ознайомимося з типами дефектів, що було знайдено під час автоматизації, та продемонструємо їх кількість. Для цього розробимо наступний код на мові програмування Python:

Прог. 1. Код для побудова кругової діаграми загальної кількості знайдених багів

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
file_path = ‘TestResults.csv’
data = pd.read_csv(file_path)
total_bugs    =    data[[‘number_of_critical_bugs’,
‘number_of_high_bugs’,     ‘number_of_medium_bugs’,
‘number_of_low_bugs’]].sum()
plt.figure(figsize=(8, 10))
plt.pie(total_bugs,        labels=total_bugs.index,
autopct=‘%1.1f%%’,     colors=[‘red’,     ‘orange’,
‘yellow’, ‘green’])
plt.title(‘Distribution of Bugs Categories’)
plt.show()
```

За допомогою даного коду ми можемо знайти та продемонструвати загальну кількість знайдених багів за допомогою автоматизованого тестування за час розробки (див.рис.1.). Дана діаграма

демонструю замовнику і всім членам команди ефективність автоматизованого тестування:

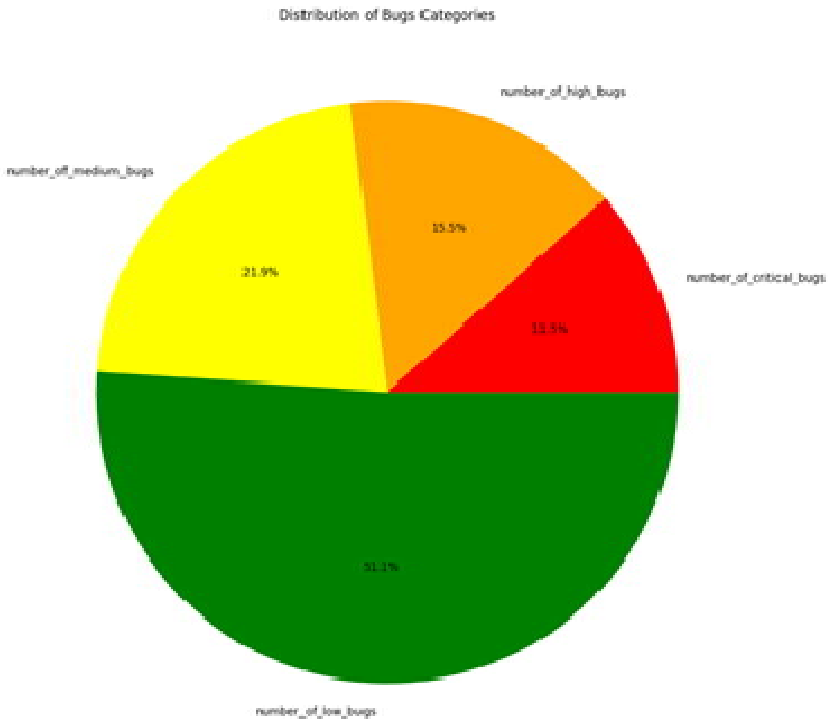

Рис. 1. Діаграма загальної кількості знайдених багів за час розробки

Далі продемонструємо тенденцію виявлення багів за допомогою автоматизованого тестування. Для цього розробимо наступний код:

Прог. 2. Код для побудови точкової гістограми тенденції виявлення багів

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
file_path = ‘TestResults.csv’
data = pd.read_csv(file_path)
plt.figure(figsize=(12, 8))
```

```
sns.pointplot(x=data['number_of_sprint'],
y=data['number_of_critical_bugs'],      color='red',
label='Critical Bugs')
sns.pointplot(x=data['number_of_sprint'],
y=data['number_of_high_bugs'],       color='orange',
label='High Bugs')
sns.pointplot(x=data['number_of_sprint'],
y=data['number_of_medium_bugs'],     color='yellow',
label='Medium Bugs')
sns.pointplot(x=data['number_of_sprint'],
y=data['number_of_low_bugs'],         color='green',
label='Low Bugs')
plt.title('Bugs Count for Each Sprint')
plt.xlabel('Sprint Number')
plt.ylabel('Number of Bugs')
plt.ylim(-1, 25)
plt.legend()
plt.show()
```

За допомогою даного коду є можливим знайти та продемонструвати тенденцію виявлення багів за допомогою автоматизованого тестування за час розробки. Можно зробити висновок, що автоматизоване тестування позитивно вплинула на розробку, з кожним спринтом було виявлено меншу кількість багів (див. рис. 2):

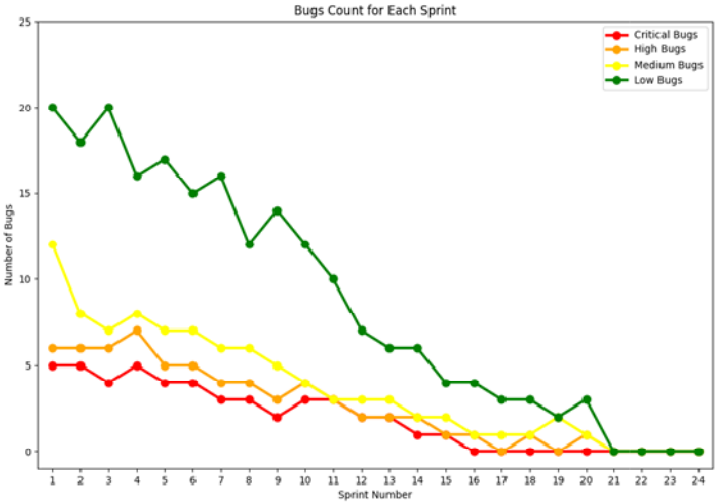

Рис. 2. Точкова гістограма тенденції виявлення багів

Далі продемонструємо залежність кількості дефектів від покриття IT-проекту автоматизованими тестами. Для цього розробляється наступний код:

Прог. 3. Код для побудови діаграми залежності кількості дефектів від покриття IT-проекту тестами

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
file_path = ‘TestResults.csv’
data = pd.read_csv(file_path)
correlation_critical =
data[‘number_of_critical_bugs’].corr(data[‘test_cov
erage’])
correlation_high =
data[‘number_of_high_bugs’].corr(data[‘test_coverag
e’])
correlation_medium =
data[‘number_of_medium_bugs’].corr(data[‘test_cover
age’])
correlation_low =
data[‘number_of_low_bugs’].corr(data[‘test_coverage
’])
plt.figure(figsize=(15, 5))
plt.subplot(1, 4, 1)
sns.lineplot(x=data[‘number_of_sprint’],
y=data[‘number_of_critical_bugs’], label=‘Critical
Bugs’, marker=‘o’)
sns.lineplot(x=data[‘number_of_sprint’],
y=data[‘test_coverage’], label=‘Test Coverage’,
marker=‘o’)
plt.title(‘Critical Bugs vs Test Coverage’)
plt.subplot(1, 4, 2)
sns.lineplot(x=data[‘number_of_sprint’],
y=data[‘number_of_high_bugs’], label=‘High Bugs’,
marker=‘o’)
sns.lineplot(x=data[‘number_of_sprint’],
y=data[‘test_coverage’], label=‘Test Coverage’,
marker=‘o’)
plt.title(‘High Bugs vs Test Coverage’)
plt.subplot(1, 4, 3)
```

```
sns.lineplot(x=data['number_of_sprint'],
y=data['number_of_medium_bugs'], label='Medium
Bugs', marker='o')
sns.lineplot(x=data['number_of_sprint'],
y=data['test_coverage'], label='Test Coverage',
marker='o')
plt.title('Medium Bugs vs Test Coverage')
plt.subplot(1, 4, 4)
sns.lineplot(x=data['number_of_sprint'],
y=data['number_of_low_bugs'], label='Low Bugs',
marker='o')
sns.lineplot(x=data['number_of_sprint'],
y=data['test_coverage'], label='Test Coverage',
marker='o')
plt.title('Low Bugs vs Test Coverage')
plt.tight_layout()
plt.show()
```

Проаналізувавши діаграму, яка наведена на рис.3, можно зробити висновок, що присутня сильна обернена кореляція — чим більше тестів заавтоматизовано, тим менше багів буде знайдено під час релізу:

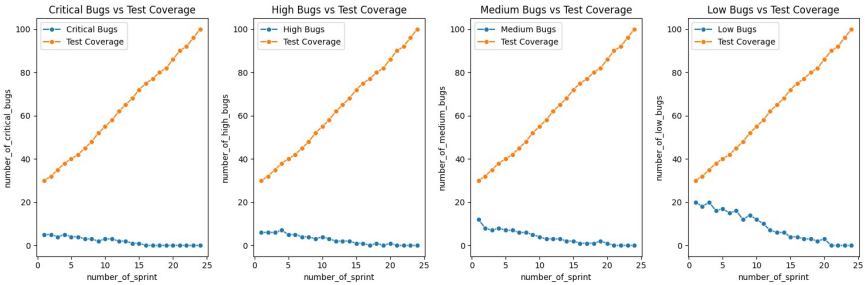

Рис. 3. Діаграма залежності кількості дефектів від покриття IT-проекту тестами

Спираючись на код та отримані діаграми, аналіз результатів виявив дуже позитивний вплив автоматизованого тестування на розробку IT-проекту. Автоматизоване тестування врази зменшило кількість дефектів та пришвидшило розробку IT-проекту.

Висновки. За допомогою отриманих результатів аналізу даних, є можливим визначити, що автоматизоване тестування дуже

позитивно вплинуло на розробку IT-проекту: покращило його якість та зменшило час розробки. IT-проект не потребую оптимізації процесів тестування та може бути взятим у ролі прикладу для майбутніх IT-проектів.

Автоматизоване тестування відіграє визначну роль у поліпшенні якості програмного забезпечення, скороченні термінів розробки та зниженні витрат на неї. Зокрема, автоматизація тестування дозволяє ефективно виконувати повторювані тестові сценарії, виявляти помилки на ранніх стадіях розробки та забезпечує швидке виправлення дефектів.

У загальному, можна дійти до висновку, що використання автоматизованого тестування є необхідною складовою сучасних IT-проектів, для досягнення високої якості програмного забезпечення та ефективного управління.

Бібліографічні посилання

1. Pargaonkar, S. (2023). A Comprehensive Research Analysis of Software Development Life Cycle (SDLC) Agile & Waterfall Model Advantages, Disadvantages, and Application Suitability in Software Quality Engineering. International Journal of Scientific and Research Publications (IJSRP), 13(08).
2. Роберта К. Мартіна, Джеймса В. Ньюкірка, Роберта С. К. Швидка розробка програм. Принципи, приклади, практика. 2004, 752 с.
3. Аксельрод А. Повний посібник з автоматизованого тестування. Springer Science+Business Media Нью-Йорк, 2018. 542 с.
4. Говорущенко Т.О., Малярчук Р.А. Аналіз сучасних методологій розробки складних програмних проєктів. Хмельницький: ХНУ, 2014. 219 с.
5. Cockburn A. Agile software development. — Boston : Addison-Wesley, 2002. 177 p.
6. Sommerville I. Software Engineering: United States Edition / I. Sommerville, 9 edition., Boston: Pearson, 2010. 792 p.
7. Cole R. Scotcher E. Agile project management. United Kingdom: Pearson Education Limited, 2015. 198 p.
8. What is Automation Testing: Benefits, Strategy, Tools. URL: https://www.browserstack.com/guide/automation-testing-tutorial.
9. What Is Automation Testing? Ultimate Guide & Best Practices. URL: https://katalon.com/resources-center/blog/what-is-automation-testing.
10. Automated software testing. URL: https://www.atlassian.com/continuous-delivery/software-testing/automated-testing.
11. Software Testing — Automation Testing. URL: https://www.geeksforgeeks.org/software-testing-automation-testing/.

12. Data Analysis with Python. URL: https://www.freecodecamp.org/learn/data-analysis-with-python/.
13. Python Programming Tutorials URL: https://pythonprogramming.net/data-analysis-tutorials/.
14. Data analysis using Pandas. URL: https://www.geeksforgeeks.org/python-data-analysis-using-pandas/.
15. Data Visualization with Python Seaborn. URL: https://www.geeksforgeeks.org/data-visualization-with-python-seaborn/.
16. How to use Seaborn for Data Visualization. URL: https://www.section.io/engineering-education/seaborn-tutorial/.
17. J. Wang, Y. Huang, C. Chen, Z. Liu, S. Wang and Q. Wang, «Software Testing with Large Language Models: Survey, Landscape, and Vision,» (2024) in IEEE Transactions on Software Engineering, doi: 10.1109/TSE.2024.3368208.

Статтю подано до редакції: 18.11.2023