УДК 004.415.2
DOI https://doi.org/10.32782/2663-5941/2025.6.2/24

Костромін А.А.
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Голубєв Л.П.
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Ківа І.Л.
Таврійський національний університет імені В.І. Вернадського

ІНФОРМАЦІЙНА СИСТЕМА МОНІТОРИНГУ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ З ІНТЕЛЕКТУАЛЬНИМ АНАЛІЗОМ РЕЗУЛЬТАТІВ ТЕСТУВАННЯ

Стаття присвячена започаткуванням для розробки програмного забезпечення інформаційної системи моніторингу якості програмного забезпечення з інтелектуальним аналізом результатів тестування.

У статті розглянуто інтеграцію сучасних методів аналізу логів у процес автоматизованого тестування програмного забезпечення. Основну увагу приділено порівнянню підходів скриптового аналізу логів із застосуванням бібліотек pytpy та scipy.stats і інтелектуального аналізу із використанням великих мовних моделей llama. Розроблений прототип системи успішно інтегрований в пайплайн Jenkins, що дозволяє автоматично виконувати тести, обробляти результати та генерувати висновки й рекомендації для кінцевих користувачів.

Скриптовий аналіз забезпечує швидкість і надійність обробки логів, дозволяючи швидко отримувати базові метрики тестування, такі як час виконання тестів, кількість успішних, провалених та пропущених тестів, а також виявляти аномалії. Використання регулярних виразів та статистичних методів дозволяє автоматизувати цей процес і виділяти основні проблеми у виконанні тестів.

Інтелектуальний аналіз, у свою чергу, додає додатковий рівень глибини та інтерпретації даних. Завдяки використанню моделей машинного навчання, таких як llama, система може класифікувати помилки, виявляти системні проблеми, створювати текстові рекомендації для покращення тестового середовища та прогнозувати ризики. Це дозволяє системі не лише виявляти поточні проблеми, але й пропонувати шляхи їх вирішення.

Окрім того, стаття акцентує увагу на перевагах інтеграції такої системи в CI/CD процеси. Це дозволяє забезпечити безперервний зворотній зв'язок для команди тестувальників та розробників, дозволяючи автоматично реагувати на критичні помилки та оптимізувати процес тестування. Генеровані рекомендації включають покращення використання програмного коду, оптимізацію часу підготовки тестів, перехід на більш надійні методи та моніторинг ресурсів для уникнення витоків.

Запропонована система є гнучкою та легко адаптується до потреб різних проектів. Вона може бути розширена для підтримки інших бібліотек тестування, інтеграції з хмарними технологіями для зберігання та аналізу логів або використання більш складних моделей ШІ. Таким чином, запропонована система є ефективним рішенням для забезпечення високої якості програмного забезпечення в умовах сучасних вимог до автоматизації тестування.

Ключові слова: *автоматизоване тестування, автоматизація процесів, якість програмного забезпечення, інформаційна системи, штучний інтелект.*

Постановка проблеми. Сучасні інформаційні системи для моніторингу якості програмного забезпечення стикаються з низкою проблем, пов'язаних із обробкою великих обсягів даних, що генеруються в процесі тестування. Значна кількість популярних інструментів мають обмежений функціонал для глибокого аналізу результатів тестування, а також часто характеризуються незручним інтерфейсом користувача. Це знижує швидкість і точність прийняття рішень розроб-

никами, які займаються усуненням виявлених помилок. Зокрема, аналіз результатів тестів часто зводиться до ручного перегляду даних без можливості автоматичного виявлення закономірностей чи прогнозування потенційних проблем.

З іншого боку, розвиток сучасних технологій, таких як штучний інтелект та машинне навчання, відкриває нові можливості для автоматизації аналізу результатів тестування. Використання інтелектуальних алгоритмів дозволяє не лише знаходити приховані дефекти в програмному забезпеченні, а й прогнозувати їх появу, швидко класифікувати та надавати рекомендації, оптимізуючи тим самим процес розробки. Отже, інтеграція таких інновацій у системи моніторингу якості є важливим завданням.

Важливість даної проблематики підтверджується потребою у вдосконаленні процесів забезпечення якості програмного забезпечення. У сучасному світі, де програмні продукти стають дедалі складнішими, а їх якість – критично важливою. Створення зручних, ефективних і функціональних інформаційних систем для автоматизації тестування та аналізу результатів стає нагальною потребою. Виходячи з цього, розробка таких систем із залученням штучного інтелекту є дуже актуальною темою сьогодення.

Забезпечення якості ПЗ базується на постійному моніторингу результатів тестування та впровадженні інтелектуальних методів їх аналізу для своєчасного виявлення дефектів і підвищення надійності системи в цілому. При цьому інформаційні системи моніторингу якості повинні бути інтегрованими в процес CI/CD, автоматизуючи збір і обробку результатів тестових прогонів і надаючи зручні звіти.

Аналіз останніх досліджень і публікацій. У сучасному світі забезпечення якості програмного забезпечення (ПЗ) відіграє важливу роль, а зростаючі обсяги даних, отриманих в процесі тестування, вимагають нових підходів до їх обробки. За останні роки з'явився ряд рішень, які інтегрують штучний інтелект (ШІ) і машинне навчання (МН) у процеси моніторингу якості ПЗ, зокрема для автоматизованого аналізу результатів тестування. Основна ідея таких систем полягає не лише у зборі метрик (наприклад, скільки тестів пройшло/завалилось), але й у використанні алгоритмів ШІ для пошуку закономірностей, аномалій і прогнозів у масивах тестових даних [1-3].

Один із ключових напрямів досліджень – це створення аналітичних панелей, які дозволяють візуалізувати метрики тестування та аналізувати історичні тренди. Наприклад, сервіс Applitools Test Insights використовує ШІ для виділення ключових метрик і трендів, що дозволяє командам оцінювати якість ПЗ за допомогою аналізу історичних даних [5].

Ще одним напрямом є оптимізація тестової стратегії за допомогою ШІ. Наприклад, платформи algoQA та AlgoShack застосовують ML–моделі для аналізу зв'язків між тест–кейсами та змінами в коді. Після виконання тестів система ідентифікує тести, які найбільше корелюють із новими функціями, і автоматично запускає їх, що дозволяє уникнути повного прогону тестового набору після кожної зміни [6].

Проте, незважаючи на досягнення, у цій сфері залишаються відкритими низка питань. По–перше, якість даних, які використовуються для навчання моделей МН, є критичним фактором. Нестача репрезентативних наборів даних або невірне класифікування помилок може призводити до помилкових висновків. Крім того, однією з проблем є відсутність єдиних стандартів інтеграції інструментів. Наприклад, такі інструменти як Jira, Jenkins чи TestRail мають різні формати обміну даними, що ускладнює створення уніфікованих систем моніторингу. Також недостатньо уваги приділяється тестуванню не–функціональних характеристик, таких як продуктивність, безпека чи аналітика [7].

Ще одним важливим напрямом, який потребує подальшого вдосконалення, є адаптивність таких систем для мобільних пристроїв. Хоча більшість існуючих панелей орієнтовані на десктопні інтерфейси, зростає потреба в адаптивному дизайні, який дозволить зручно переглядати результати тестування на мобільних пристроях. Це особливо важливо для команд, які активно працюють у розподілених середовищах, де доступ до звітів із мобільного телефону може бути критично важливим.

Крім того, інтеграція з інструментами автоматизації, такими як Jenkins, є ще одним перспективним напрямом. Створення єдиної системи, яка зможе не лише запускати тестування, але й автоматично аналізувати результати та генерувати звіти у зручному для користувача форматі, значно спростить процес забезпечення якості ПЗ.

Таким чином, можна зробити висновок, що, хоча існує багато досягнень у сфері моніторингу якості ПЗ, особливу увагу слід приділити порівняння різних методів аналізу результатів тестування разом з продуктивністю й забезпеченням рекомендацій для удосконалення процесів і безпеки та інтеграції з існуючими DevOps–інстру-

ментами. Ці аспекти мають потенціал для значного покращення ефективності та зручності використання таких систем.

Постановка завдання. Метою статті є аналіз та порівняння методів тестування та автоматизації процесів тестування. Доцільність використання сучасних моделей для інтелектуального аналізу тестових результатів. Визначення вимог до архітектури системи моніторингу з інтеграцією ШІ–модулів для класифікації результатів з прогнозуванням можливих ризиків та наданням рекомендацій. Розробка прототипу інформаційної системи, що автоматично збирає дані з тестових фреймворків та виконує аналіз результатів із застосуванням великих мовних моделей. Порівняння результатів різних методів аналізу з традиційними методами аналізу тестових результатів.

Об'єктом дослідження є процес створення та експлуатації інформаційних систем моніторингу якості програмного забезпечення.

Предметом дослідження є порівняння методів та алгоритмів аналізу результатів тестування ПЗ для автоматичного виявлення та класифікації дефектів.

Виклад основного матеріалу. Запропонована система моніторингу якості програмного забезпечення побудована на інтеграції автоматизованого тестування з використанням Selenium, бібліотеки pytest, а також засобів для інтелектуального аналізу результатів тестування. Архітектура базується на пайплайні Jenkins, який забезпечує автоматичний запуск тестів, обробку результатів і подальший аналіз логів. Поглянемо на основні етапи роботи системи.

По–перше, усі функціональні тести реалізовані з використанням Selenium для перевірки веб–додатків. Для управління тестами використовується бібліотека pytest, яка дозволяє зручно структурувати тести, виконувати їх у паралельному режимі та генерувати докладні логи. Логи тестів зберігаються у визначеній директорії tests/logs для подальшого аналізу.

По–друге, Jenkins забезпечує автоматичний запуск тестів через налаштований пайплайн, що складається з наступних основних етапів:

1. Jenkins завантажує релізну гілку репозиторію, яка містить актуальні тести та конфігурації;
2. Створюється Python–віртуальне середовище, встановлюються необхідні залежності з requirements.txt;
3. Запуск серверу та виконання тестів;
4. Сервер запускається у фоновому режимі;
5. Виконуються тести за допомогою pytest, а результати записуються до лог–файлу;
6. Після завершення тестів сервер зупиняється;
7. Скриптовий аналіз результатів;
8. Інтелектуальний аналіз результатів з ШІ.

Основними точками будуть служити два методи аналізу результатів. В першому логи аналізуються скриптом, який використовує бібліотеки numpy та scipy.stats. Цей аналіз дозволяє:

− Визначити базові метрики, такі як час виконання тестів, кількість помилок тощо.

− Виконати статистичний аналіз для виявлення відхилень у результатах тестування.

− Класифікувати помилки за шаблонами для виділення ключових груп проблем.

На другому етапі здійснюється більш глибокий аналіз логів з використанням бібліотек llama та transformers. В даному випадку використовується модель open_llama_3b_code_instruct_0.1.Q8_0.gguf. Аналіз включає: За допомогою моделей машинного навчання логи класифікуються за успішними та провальними. На наступному кроці проводиться детальний аналіз тестів зі статусом Failed, Skipped де розписуються винятки та джерела помилок, а також можливі системні проблеми. На наступному кроці проводиться пошук явних та неявних залежностей в тестах для виявлення складних аномалій, які можуть пов'язувати тести. На наступному кроці проводиться аналіз ресурсів, тобто найбільший затрачений час на тестування. В кінці повторний аналіз усіх результатів та надання рекомендацій для уникнення ризиків і підвищення ефективності тестування.

Підсумовуючи вище сказане, запропонована архітектура дозволяє ефективно інтегрувати інтелектуальний аналіз у процес тестування, забезпечуючи високу якість та стабільність програмного забезпечення В свою чергу, використання пайплайнів забезпечує автоматичне тестування і аналіз логів. Подвійний аналіз дозволяє оперативно отримувати розширені результати з рекомендаціями та базовими метриками тестування.

Для створення прототипу інформаційної системи з використанням ШІ було здійснено декілька етапів: наведено опис програмних і апаратних засобів, даних та методів, які були використані для розробки інформаційної системи моніторингу якості ПЗ з інтелектуальним аналізом результатів тестування. Описано стек технологій (мова програмування, бібліотеки для використання LLM та автоматизації тестування), конфігурація пайплайнів, структуру вхідних даних і підхід до їх підготовки, методи аналізу даних.

Як основний інструмент розробки обрана мова програмування Python 3.10, що підтримує всі

необхідні бібліотеки для проведення тестування, інтелектуального аналізу даних та автоматизації [4]. Виходячи із поставленої мети, буде обрано наступні інструменти та бібліотеки:

– для машинного аналізу буде використовуватись відома бібліотека transformers для використання простих LLM моделей;

– для скриптового аналізу буде використовуватись бібліотека SciPy та NumPy для зчитування стандартної інформації та пошуку простих аномалій;

– для автоматизації UI–тестів буде використовуватись Selenium WebDriver для написання сценаріїв браузерного тестування та найпоширеніша бібліотека тестування pytest для організації та виконання автоматизованих тестів;

– для налаштування процесу автоматизації CI/CD буде використовуватись Jenkins Pipeline для автоматичного запуску збірок, тестів і деплою при оновленні коду;

– для зручного розгортання системи буде використовуватись контейнеризація за допомогою Docker для пакування кожного компонента в незалежний контейнер.

Вхідними даними будуть виступати логи результатів тестів у текстовому форматі, згенеровані Selenium та pytest. На рис. 1 зображено приклад вхідних даних для аналізу.

Для правильного масштабування буде обрана мікросервісна архітектура, яка включає принцип Single Responsibility, де кожний сервіс буде упакований у Docker–контейнер. На платформі Jenkins буде автоматично налаштовано запуск pytest–тестів та веб–сайту на сервері, скриптовий аналіз результатів тестування та інтелектуальний аналіз. На рис. 2 зображено частину вікна програмного забезпечення Jenkins з фрагментом коду для пайплайну.

В свою чергу, сам пайплайн автоматизації показаний на рис. 3, де вказано час, який був затрачений на кожному етапі.

Скриптовий аналіз відповідає за базову обробку логів. Це парсинг логів з використанням регулярних виразів для отримання ключових даних, таких як час виконання тестів та їхні статуси. Це аналіз продуктивності, де numpy забезпечує швидкий підрахунок середнього, максимального та мінімального часу для кожної фази тестів (setup, teardown, call). А також використання zscore для виявлення аномальних значень у часах виконання тестів. На рис. 4 зображено фрагмент коду для скриптового аналізу.

Фрагмент відповідає за пошук правильної комбінації символів в масиві інформації для того, щоб вилучити найкорисніші дані. На наступному рис. 5 зображено фрагмент вихідних даних скриптового аналізу.

```
head>\n<body>\n    \n    <div class="d-flex align-items-center flex-column justify-content-
center vh-100" id="header">\n        <div class="border" style="padding: 86px 250px 100px">\n
<h1 class="display-4 text-center pb-5">Login</h1>\n            <form class="" action=""
method="post">\n                <input type="hidden" name="csrfmiddlewaretoken" value="">\n
<div class="form-group">\n                    <input name="email" class="form-control form-
control-lg" placeholder="email" type="email" required="">\n                </div>\n
<div class="form-group">\n                    <input name="password" class="form-control form-
control-lg" placeholder="Password" type="password" required="">\n                </div>\n
<div class="form-group d-flex flex-row-reverse">\n                    <button class="btn btn-
primary btn-lg" style="padding: 8px 30px">Login</button>\n                </div>\n
</form>\n            \n        </div>\n    </div>\n\n\n    <script type="text/javascript"
src="/static/bootstrap/js/jquery-3.6.3.min.js"></script>\n    <script type="text/javascript"
src="/static/bootstrap/js/bootstrap.bundle.min.js"></script>\n    \n\n</body></html>'
/Users/admin/Desktop/project/env/lib/python3.13/site-packages/selenium/webdriver/remote/
errorhandler.py:232: selenium.common.exceptions.InvalidSelectorException: Message: invalid
selector: Unable to locate an element with the xpath expression //a[contains(@href,'/sign/
logout/') because of the following error:
/Users/admin/Desktop/project/env/lib/python3.13/site-packages/selenium/webdriver/remote/
errorhandler.py:232: selenium.common.exceptions.NoSuchElementException: Message: no such
element: Unable to locate element: {"method":"xpath","selector":"//td[contains(text(), 'Новий
предмет')]"}
============================== slowest durations ==============================
9.21s setup    tests/selenium_tests/auth/test_login_negative.py::test_failed_login
5.53s setup    tests/selenium_tests/auth/test_logout.py::test_logout
5.37s setup    tests/selenium_tests/common/test_create_new_item.py::test_authorized_page_access
5.29s setup    tests/selenium_tests/common/test_create_new_item.py::test_create_new_item
2.49s call     tests/selenium_tests/auth/test_login_positive.py::test_successful_login
0.95s setup    tests/selenium_tests/auth/test_login_positive.py::test_successful_login
0.35s call     tests/selenium_tests/auth/test_login_negative.py::test_failed_login
0.17s teardown tests/selenium_tests/common/test_create_new_item.py::test_authorized_page_access
0.16s call     tests/selenium_tests/common/test_create_new_item.py::test_create_new_item
0.16s teardown tests/selenium_tests/common/test_create_new_item.py::test_create_new_item
0.13s teardown tests/selenium_tests/auth/test_login_negative.py::test_failed_login
0.10s teardown tests/selenium_tests/auth/test_logout.py::test_logout
0.06s call     tests/selenium_tests/auth/test_logout.py::test_logout
0.05s call     tests/selenium_tests/common/test_create_new_item.py::test_authorized_page_access
0.01s teardown tests/selenium_tests/auth/test_login_positive.py::test_successful_login
======================== 3 failed, 2 passed in 30.06s =========================
```

Рис. 1. Приклад вхідних даних

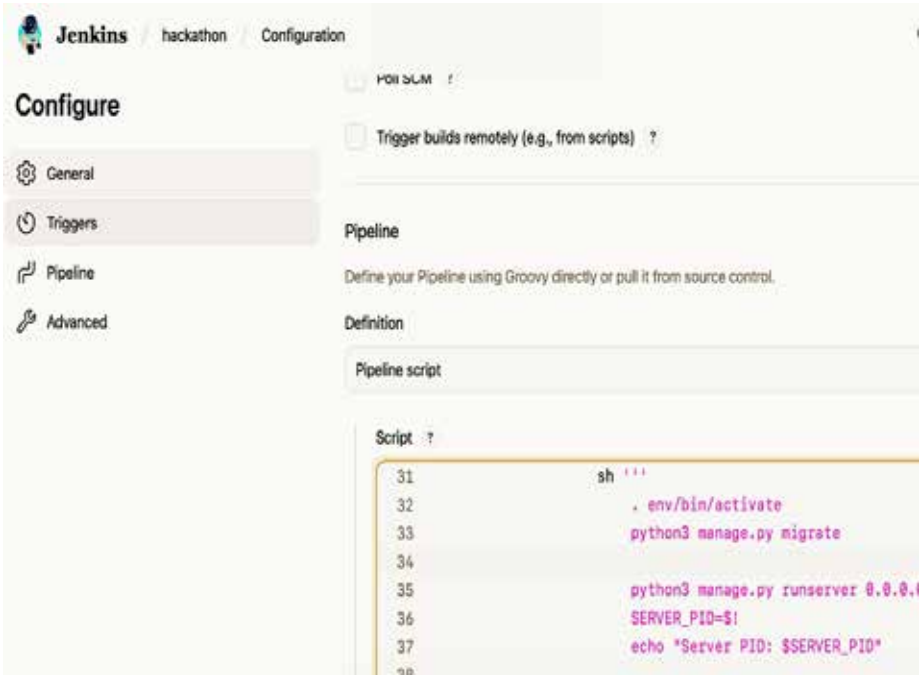

Рис. 2. Приклад програмного забезпечення Jenkins

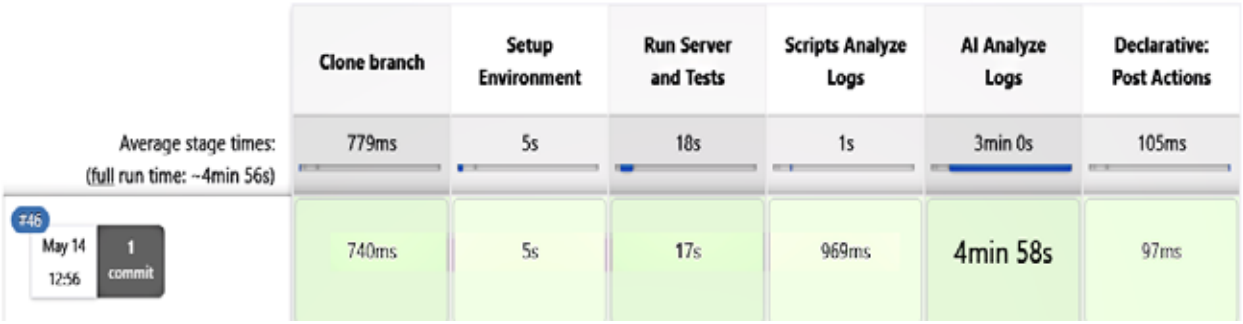

Рис. 3. Пайплайн автоматизації

```python
def parse_test_logs(logs: list[str]):
    time_pattern = re.compile(r"(\d+\.\d+)s (setup|teardown|call)\s+([\w/:\.]+)")
    status_pattern = re.compile(r"(=\s+)(\d+ failed, \d+ passed)")

    test_times = []
    statuses = []

    for line in logs:
        time_match = re.search(time_pattern, line)
        if time_match:
            time = float(time_match.group(1))
            phase = time_match.group(2)
            test_name = time_match.group(3)
            test_times.append((test_name, phase, time))

        status_match = re.search(status_pattern, line)
        if status_match:
            statuses.append(status_match.group(2))

    return test_times, statuses
```

Рис. 4. Фрагмент коду для скриптового аналізу

```
Аналіз часу виконання:

Фаза: setup
  Кількість виконань: 5
  Середній час: 5.27s
  Максимальний час: 9.21s
  Мінімальний час: 0.95s

Фаза: teardown
  Кількість виконань: 5
  Середній час: 0.11s
  Максимальний час: 0.17s
  Мінімальний час: 0.01s

Фаза: call
  Кількість виконань: 5
  Середній час: 0.62s
  Максимальний час: 2.49s
  Мінімальний час: 0.05s

Аномалії:
  Тест: tests/selenium_tests/auth/test_login_negative.py::test_failed_login, Фаза: setup, Час: 9.21s
```

Рис. 5. Фрагмент вихідних даних для скриптового аналізу

```python
def llama_analysis(logs: str, classification: str) -> str:
    prompt = f"""
      Analyze the following test logs and provide a short summary:\n\n{logs}\nOverall classification:
  {classification}\n\nSummary:
    """
    inputs = tokenizer(prompt, return_tensors="pt")
    outputs = model.generate(inputs.input_ids, max_length=1000)
    summary = tokenizer.decode(outputs[0])
    return summary
```

Рис. 6. Фрагмент коду для інтелектуального аналізу

Як бачимо, час на аналіз виявився дуже малим. Всього лише 969 мс., що свідчить про гарну швидкість коду. Зауважимо, що усі дані перевіряються з строго визначеною структурою. У випадку, якщо вхідні дані змінять свою структуру, то скрипт не зможе провалідувати та проаналізувати дані.

Проаналізувавши результати, можемо виявити наступні переваги та обмеження скриптового підходу:

Переваги:

– Висока швидкість виконання;
– Простота реалізації;
– Можливість отримання базових метрик без надлишкових обчислень.

Обмеження:

– Немає можливості генерувати глибокі висновки чи рекомендації;
– Залежність від заздалегідь визначених правил.

Аналіз з використанням штучного інтелекту відповідає за більш глибку перевірку логів на основі алгоритмів та методів великих мовних моделей. В процесі обробки виконується класифікацію тестів для визначення загальних тенденцій. Також проводиться детальний аналіз тестів, що не пройшли з різних причин, де описуються винятки та джерела помилок. Паралельно проводиться пошук явних та неявних залежностей для виявлення аномалій та транзитивних залежностей. Додатково аналізуються тести, які зайняли найбільше ресурсів та часу на тестування. В кінці ШІ підсумовує результати та надає рекомендацій для підвищення ефективності тестування. На рис. 6 зображено фрагмент коду, який відповідає за використання відповідних бібліотек для інтелектуального аналізу.

На цьому прикладі ми бачимо, що код виглядає не сильно важким та об'ємним для розуміння, бо вся логіка криється всередині моделі. На рис. 7 зображено фрагмент результатів інтелектуального аналізу. Він виявився набагато більшим та розгорнутим ніж його попередник.

```
**Test Outcomes & Patterns:**

    * **Passed tests:** `test_failed_login`, `test_successful_login`,
`test_authorized_page_access` (3 tests)
    * **Failed tests:** `test_logout`, `test_create_new_item` (2 tests)
    * **Skipped tests:** None
    * **Erroneous tests:** None

    There are no flaky tests identified in this test session, as each test has a consistent
outcome. Failures are isolated to specific modules, specifically `tests/selenium_tests/auth/
test_logout.py` and `tests/selenium_tests/common/test_create_new_item.py`.

    **Test Dependencies:**

    * There are no explicit dependencies between tests identified in this test session.
    * However, the use of Selenium WebDriver and the shared resources (e.g., browser sessions)
may introduce implicit dependencies between tests.

    **Error Analysis:**

    * **Exceptions:**
        + `InvalidSelectorException`: Unable to locate an element with the XPath expression `//
a[contains(@href,'/sign/logout/')]`
        + `NoSuchElementException`: Unable to locate element with the XPath expression `//
td[contains(text(), 'Новий предмет')]`
    * **Error origins:**
        + The `InvalidSelectorException` originates from the `test_logout` test, specifically
from the line where the test attempts to locate the logout link.
        + The `NoSuchElementException` originates from the `test_create_new_item` test,
specifically from the line where the test attempts to locate the element with the text "Новий
предмет".
```

Рис. 7. Приклад результатів інтелектуального аналізу

В даному випадку, час на аналіз виявився навпаки – великим. Майже 5 хвилин, що свідчить про повільне виконання коду. Зауважимо, що час може бути набагато більшим, через великі обчислення моделі. Для прикладу використовувалась модель open_llama_3b_code_instruct_0.1.Q8_0.gguf, яка має 3 мільярди параметрів та 8–бітне квантування, що зменшує розмір моделі, але може знижувати точність і збільшувати час інференсу [8] на слабкому залізі. Це особливо критично при відсутності GPU–прискорення або обмеженій оперативній пам'яті. Тут криється багато підводних каменів, які одразу не кидаються в очі. Наприклад, якщо для обчислень використовувати CPU (процесор) замість GPU (відеокарти), це змушує модель виконувати матричні операції послідовно, що різко збільшує затримки. Тож потрібно досить потужне обладнання з декількома GPU для швидкого обрахунку відповіді.

Проаналізувавши результати, можемо виявити наступні переваги та обмеження аналізу штучного інтелекту:

Переваги:

− Можливість створення більш глибоких інсайтів;

− Генерація текстових висновків та рекомендацій;

− Аналіз від апаратної до програмної частини;

− Гнучкість у навчанні моделей для нових типів даних;

− Має усі можливості скриптового підходу та сильно його перевершує.

Обмеження:

− Набагато більша обчислювальна складність порівняно зі скриптовим аналізом;

− Сильна залежність від апаратних можливостей системи, на якій буде запускатись інфраструктура.

− Залежність від якості вхідних даних і попередньої класифікації.

Висновки. У даному дослідженні розглянуто інтеграцію сучасних методів аналізу результатів автоматизованого тестування програмного забезпечення. Розроблено прототип системи що базується на комбінації скриптового аналізу та інтелектуального аналізу із застосуванням аналітичних та спеціалізованих бібліотек для аналітики даних та генерації висновків за допомогою великих мовних моделей. Система забезпечує автоматичний збір логів під час виконання тестів та їх подальший аналіз у кілька етапів. Результати інтегруються в CI/CD процес, забезпечуючи зворотний зв'язок для команд розробників і тестувальників. Запропонована архітектура системи дозволяє легко адаптувати алгоритми аналізу до нових сценаріїв, розширювати можливості ШІ–моделей та інтегрувати нові інструменти для подальшої взаємодії з результатами.

Подальший розвиток системи може включати автоматичне звітування на основі результатів аналізу. До прикладу, запуск тестів та їх аналіз у разі вияв-

лення оновлення програмного забезпечення розробниками. Також подальша розробка може включати розширене програмне забезпечення зі зручним інтерфейсом для кінцевого користувача як менеджера, тестувальника, розробника, тощо. Доцільно також буде використання хмарних сервісів для зберігання, обробки та візуалізації результатів, що дозволить масштабувати систему для великих проєктів.

Список літератури:

1. Козаченко І.В., Кисельов, Г.Д. Автоматизація тестування програмних застосунків з використанням методів машинного навчання, С.10–11 https://ela.kpi.ua/items/c09f9b0f-39c8-4c88-a7e1-571b7576c0d2 (application date: 22.04.2025).

2. What is AI Software Testing? Sujatha R: website URL: https://www.digitalocean.com/resources/articles/ai-software-testing (application date: 25.04.2025).

3. Smart Test Execution and Analysis of Results. AlgoShack: website URL: https://medium.com/%40algoshack/smart-test-execution-and-analysis-of-results-5eba88919056 (application date: 27.04.2025).

4. Official Python documentation: website URL: https://www.python.org/doc/ (application date: 28.04.2025).

5. Applitools Documentation, website URL: https://applitools.com/docs/ (application date: 02.05.2025).

6. Як за допомогою тестів пришвидшити реліз, website URL: https://dou.ua/lenta/articles/how-testing-speed-up-release/ (application date: 06.05.2025).

7. Японська генеративна модель AI Scientist спробувала змінити власний код, website URL: https://cikavosti.com/yaponska-generatyvna-model-ai-scientist-sprobuvala-zminyty-vlasnyj-kod/ (application date: 10.05.2025).

8. AI Reliability Engineering – третя епоха SRE, website URL: https://dou.ua/forums/topic/53359/ (application date: 12.05.2025).

Kostromin A.A., Holubiev L.P., Kiva I.L. INFORMATION SYSTEM FOR MONITORING SOFTWARE QUALITY WITH INTELLIGENT ANALYSIS OF TEST RESULTS

The article is devoted to the introduction of an information system for monitoring software quality with intelligent analysis of test results for software development.

The article considers the integration of modern log analysis methods into the process of automated software testing. The main attention is paid to the comparison of approaches to scripted log analysis using the numpy and scipy.stats libraries and intelligent analysis using large llama language models. The developed prototype of the system has been successfully integrated into the Jenkins pipeline, which allows you to automatically run tests, process results, and generate conclusions and recommendations for end users.

Script analysis provides speed and reliability of log processing, allowing you to quickly obtain basic testing metrics, such as test execution time, the number of successful, failed, and missed tests, as well as identify anomalies. The use of regular expressions and statistical methods allows you to automate this process and highlight the main problems in the execution of tests.

Intelligent analysis, in turn, adds an additional level of depth and interpretation of data. Thanks to the use of machine learning models, such as llama, the system can classify errors, detect system problems, create text recommendations for improving the test environment and predict risks. This allows the system not only to detect current problems, but also to suggest ways to solve them.

In addition, the article focuses on the advantages of integrating such a system into CI/CD processes. This allows you to provide continuous feedback for the team of testers and developers, allowing you to automatically respond to critical errors and optimize the testing process. The generated recommendations include improving the use of program code, optimizing test preparation time, switching to more reliable methods and monitoring resources to avoid leaks.

The proposed system is flexible and easily adaptable to the needs of different projects. It can be extended to support other testing libraries, integrate with cloud technologies for storing and analyzing logs, or use more complex AI models. Thus, the proposed system is an effective solution for ensuring high software quality in the conditions of modern requirements for test automation.

Key words: *automated testing, process automation, software quality, information systems, artificial intelligence.*

Дата надходження статті: 15.11.2025
Дата прийняття статті: 03.12.2025
Опубліковано: 30.12.2025