УДК 004.415.532

Германов І. Р., Сердюк О. О.

ПІДВИЩЕННЯ ПРОДУКТИВНОСТІ ТЕСТУВАННЯ ВЕБ-ДОДАТКІВ ШЛЯХОМ РОЗРОБКИ СИСТЕМИ АВТОМАТИЗАЦІЇ

Сучасні веб-додатки являють собою складні, багатокомпонентні системи [1], що розробляються командою або декількома командами розробників, які мають різну кваліфікацію, досвід та відповідальність. Помилка в одній частині системи може спричинити помилку в іншій частині. Такі помилки можуть коштувати дуже дорого, тому при розроблюванні вебдодатків обов‘язково здійснюється їх тестування [2], яке направлено на виявлення та усунення якомога більшої кількості помилок і таким чином на підвищення якості програмного забезпечення [3].

Однак необхідність проведення тестування програмного додатку призводить до збільшення строків виконання проекту та потребує додаткових фінансових витрат. По деяким даним [4] витрати на виявлення та виправлення помилок у розроблюваних програмах досягають до 80 % витрат на розробку проекту. Тому підвищення продуктивності тестування являється актуальною проблемою.

Для визначення функцій процесу тестування, які можуть бути автоматизовані програмними засобами, необхідна функціональна модель існуючого процесу тестування. Для побудови моделі використана методологія функціонального моделювання IDEF0. На рис. 1 показана діаграма IDEF0 існуючого процесу тестування AS-IS («як є»), яка складається із чотирьох блоків, що відповідають певним функціям процесу. Блоки взаємодіють між собою і з навколишнім середовищем за допомогою інтерфейсів, представлених стрілками.

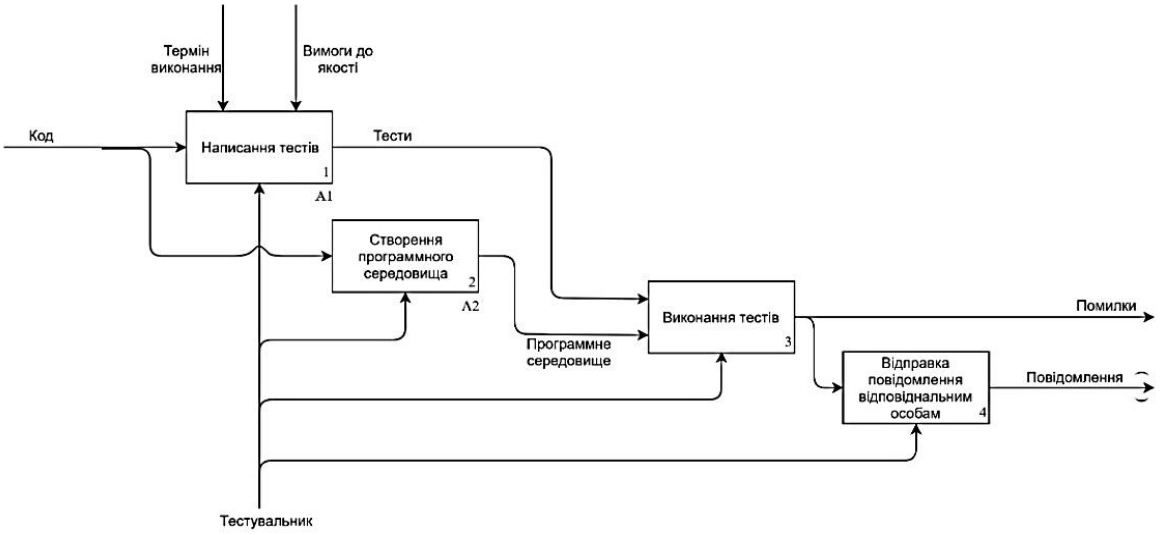

Рис. 1. Функціональна модель процесу тестування «як є»

Із рис. 1 видно, що тестувальник здійснює написання тестів, створення програмного середовища для їх виконання, запуск виконання тестів, а також підготовку і відправку повідомлень про результати тестування відповідним особам. При цьому для написання тестів і створення програмного середовища тестувальник повинен проаналізувати програмний код, а також задовольнити вимоги до якості веб-додатку при дотриманні встановленого терміну розробки проекту.

Слід зазначити, що у процесі тестування найбільш складними являються процеси написання тестів та створення програмного середовища. Тому ці процеси потребують подальшої декомпозиції.

На рис. 2 наведена функціональна діаграма процесу написання тестів.

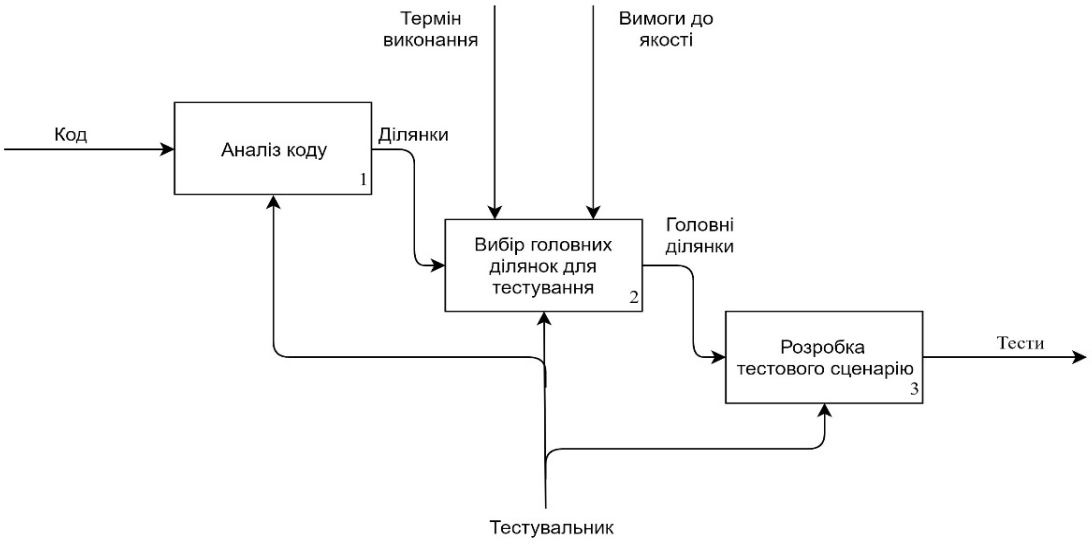

Рис. 2. Функціональна діаграма процесу написання тестів

У процесі написання тестів дуже важливою операцією являється аналіз програмного коду на предмет виділення в ньому тих ділянок, які мають найбільший вплив на можливі збитки. Після виділення цих ділянок тестувальник повинен розробити тестовий сценарій, в якому встановлюється послідовність тестування ділянок програмного коду, які можуть задіяти користувачі веб-додатку при правильних, або неправильних діях.

Недоліком в існуючій організації написання тестів являється те, що не тестуються всі ділянки коду.

На рис. 3 наведена функціональна діаграма процесу створення програмного середовища.

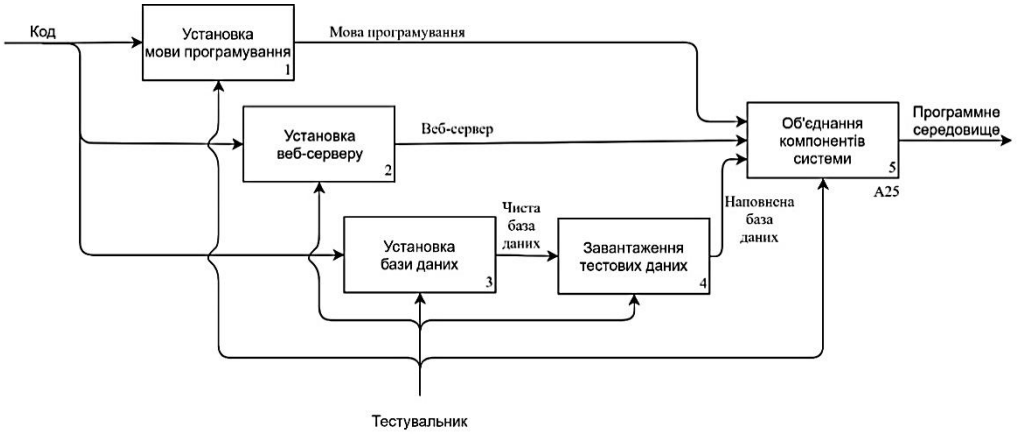

Рис. 3. Функціональна діаграма процесу створення програмного середовища

Розроблювачі веб-додатків можуть використовувати різні мови програмування. Тому для кожної мови повинне бути створене своє програмне середовище, яке розміщується на сервері. Для зберігання службової та поточної інформації створюється база даних.

Недоліком такої організації являється великий об‘єм рутинних процедур по установці і налагодженню необхідних компонентів системи, які можна автоматизувати.

В даний час рішення проблеми підвищення продуктивності тестування знаходиться на стадії дослідження. На основі останніх публікацій можна виділити декілька рішень. Для тестування інтерфейсів використовуються спеціальні інструменти для запису послідовності дій, на основі яких генеруються тестові сценарії [5]. Завдяки цим інструментам час на написання тестового сценарію зменшується. Для описання доступу до веб-сервісу іноді використовується спеціальна мова WSDL. Така мова достатньо стандартизована для того, щоб на її основі генерувати тести [6], але це рішення підходить лише для тестування компонентів системи, тому що мова WSDL описує входи та виходи окремих компонентів і не розкриває зв'язок між ними. Також ведуться дослідження у напрямку аналізу первинного коду для генерації тестів [7], але такі тести є досить примітивними і нагадують статичний аналіз коду. Генерація тестів ускладнюється також тим, що сучасні веб-додатки пишуться на динамічних мовах, в яких тип змінних визначається лише на етапі виконання.

Таким чином проведені дослідження вирішують лише деякі задачі написання тестів, але залишають невирішеними наступні задачі:

– створення програмного середовища;

– запуск тестів після кожної зміни первинного коду;

– фіксування помилок;

– відправка повідомлень.

Метою роботи є підвищення продуктивності тестування шляхом автоматизації ручних дій. Для досягнення цієї мети необхідно вирішити наступні задачі:

– автоматичне створення програмного середовища;

– автоматичний запуск тестів після кожної зміни первинного коду;

– фіксування помилок в базі даних;

– формування звіту і відправка повідомлень.

З огляду на поставлені задачі функціональна модель автоматизованого процесу тестування, тобто модель ТО ВЕ («як повинно бути»), представлена на рис. 4. Вона відрізняється від моделі «як є» тим, що функції створення програмного середовища, виконання тестів, реєстрація помилок та відправка повідомлень покладаються на автоматичну систему керування. Тестувальнику залишається тільки написання тестів для тестування виділених ділянок програмного коду, яке складно автоматизувати.

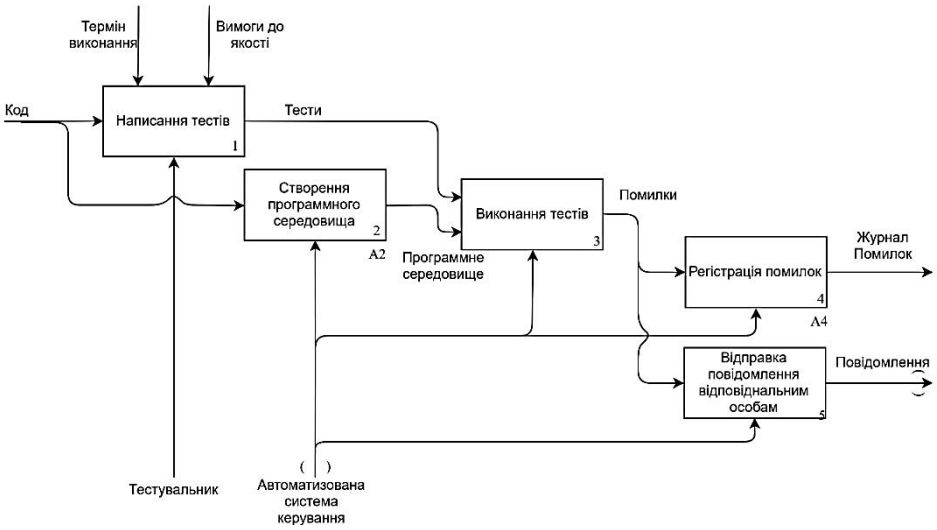

Рис. 4. Функціональна модель автоматизованого процесу тестування

Програмне середовище створюється системою автоматично на основі спеціального конфігураційного файлу, в якому розробники, фіксують всі необхідні компоненти, їх версії, та порядок запуску. Також автоматично відбувається виконання тестів після кожної зміни

програмного коду, помилки записуються в базу даних і передаються у журнал помилок, а повідомлення відправляються одразу після фіксації помилки незалежно від бажань тестувальника.

На рис. 5 представлена більш детальна модель процесу створення програмного середовища. Автоматизована система читає конфігураційний файл, який є частиною коду, і за допомогою технології контейнеризації [8], створює контейнер з мовою програмування, веб-сервером і базою даних, після чого завантажує у базу тестові дані. Для створення програмного середовища система об'єднує всі ці компоненти і налагоджує їх на виконання тестового сценарію.

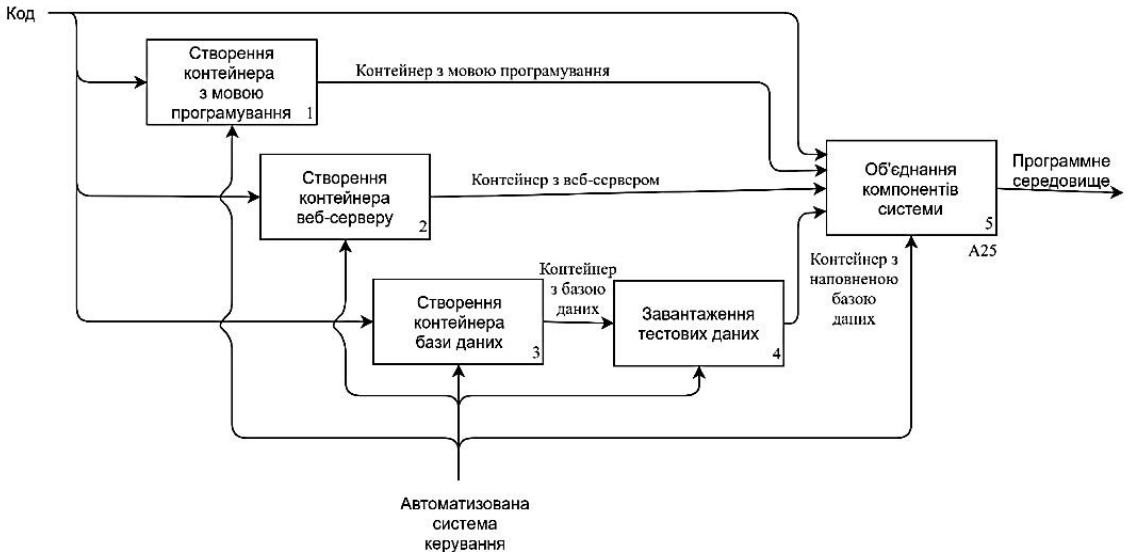

Рис. 5. Деталізована модель процесу створення програмного середовища

Модель завершального процесу автоматизованого тестування наведена на рис. 6. Система на основі виявленої помилки формує звіт і додає його до журналу.

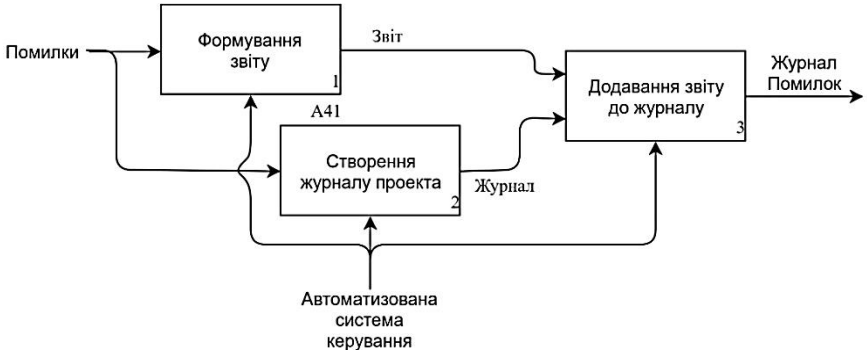

Рис. 6. Деталізована модель процесу реєстрації помилок

Для управління автоматизованою системою тестування потрібно створити програмний додаток, який завантажується потім на персональний комп'ютер тестувальника. Варіанти використання цього додатка тестувальником і програмістом (при необхідності) показані у вигляді UML-моделі на рис. 7. Як видно із діаграми, тестувальнику доступні два варіанти використання системи. Перший – це тестування проекту, що включає в себе введення тестового сценарію, а також запуск тестування, яке може виконуватися як вручну так і автоматично. Після завершення тесту можна переглянути результати цього тестування. Другий варіант – це перегляд журналу помилок, що являє собою перелік результатів тестувань.

Рис. 7. Діаграма варіантів використання системи автоматичного тестування

При виконанні процесів тестування програма системи управління процесами тестування повинна взаємодіяти із базою даних. Для моделювання бази даних побудована ER-діаграма, яка представлена на рис. 8. В базі даних необхідно зберігати такі сутності: проект, тестувальник, тест, тип тесту, помилка, тип помилки. Звіт про помилки залежить від типу помилки, типу проекту, типу теста, тому для нормалізації бази даних, він повинен формуватися динамічно. Проект і тест мають залежність один до багатьох тому, що в проекті повинно існувати більше одного тесту. Тестувальник може написати більше одного тесту, тому він також має відношення один до багатьох для таблиці тестів. Тест має лише один тип, тому таблиця тестів має відношення один до багатьох з типами тестів. Тести знаходять помилку одну, або декілька, тому тести мають відношення один до багатьох з таблицею помилок. Помилки мають різні типи, що їх характеризують, тому таблиця типів помилок має відношення один до багатьох із таблицею помилок.

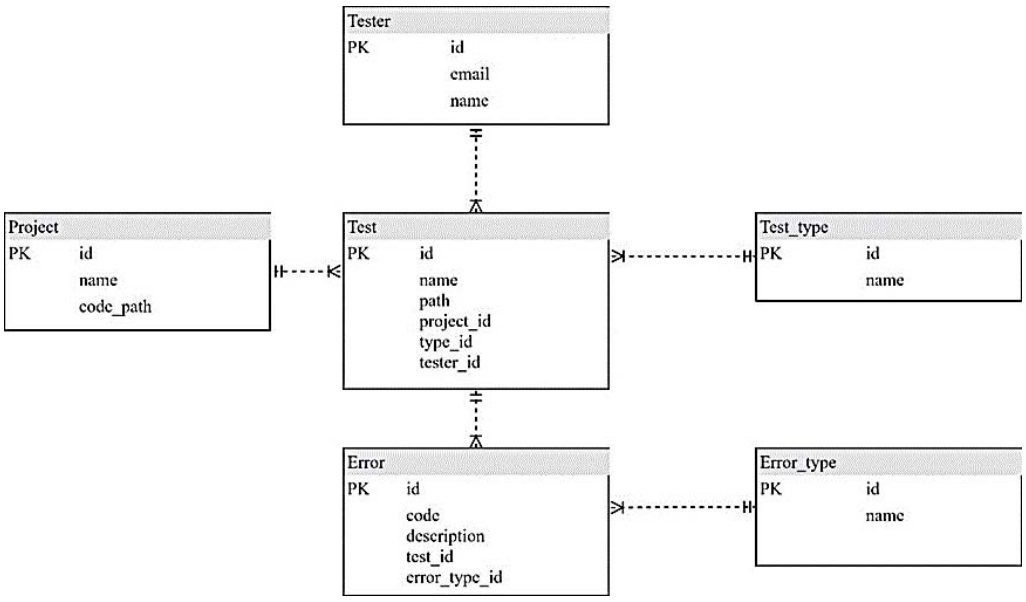

Рис. 8. ER-діаграма бази даних

Сучасні технології програмування використовують моделювання програм. Для моделювання програмного додатку побудовано UML-діаграму класів (рис. 9).

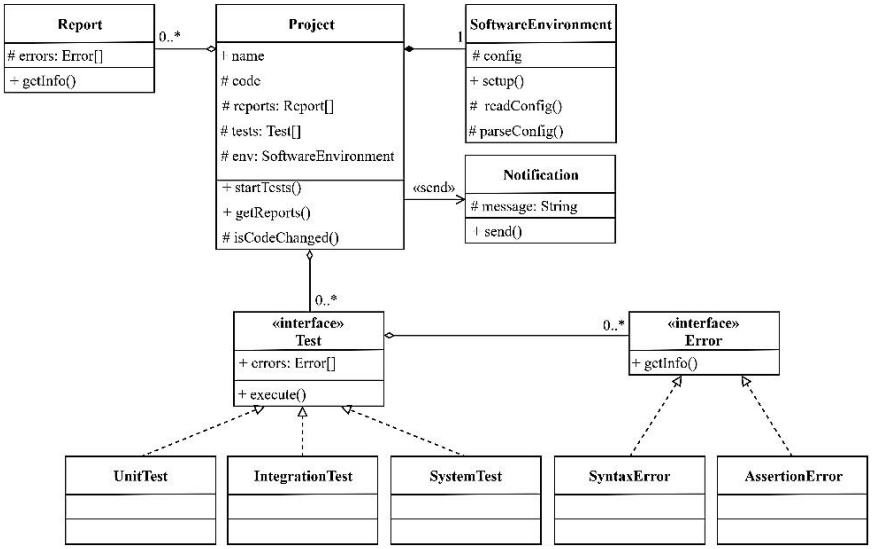

Рис. 8. Діаграма класів системи

Центральним класом є клас Project, який створює звіти, тести та програмне середовище. Проект має методи для запуску тестів, отримання звітів та перевірки змін коду у веб-додатку. Проект має лише одне програмне середовище. Клас програмного середовища має один публічний метод для установки всіх компонентів. Тести мають єдиний інтерфейс, в якому є метод запуску та атрибут з помилками. Тести мають нуль або більше помилок. Помилки мають єдиний інтерфейс з одним методом отримання інформації.

ВИСНОВКИ

В роботі засобами функціонального моделювання існуючих процесів тестування веб-додатків встановлені функції тестувальника, які потребують значних витрат часу і які доцільно автоматизувати. Розроблена функціональна модель нового бізнес-процесу («як повинно бути»), в якому передбачено звільнення тестувальника від рутинних операцій. Для реалізації автоматизованого процесу тестування розроблені моделі варіантів використання автоматизованої системи управління тестуванням, та модель бази даних. Запропонована система забезпечує підвищення продуктивності тестування веб-додатків.

Серед подальших напрямків робіт найцікавішим є генерація тестових сценаріїв на основі специфікації вимог.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Леон Ш. Архитектура веб-приложений / Ш. Леон, Р. Розен. – М. : Эксмо, 2011. – 634 с.

2. Мартюков А. С. Методология функционального тестирования программного обеспечения на основе сценариев использования и схемы приоритетов / А. С. Мартюков // Новые информационные технологии в автоматизированных системах. – 2010. – № 13. – С. 183–185.

3. Бирюков С. В. Анализ стратегий тестирования программного обеспечения / С. В. Бирюков // Известия ЮФУ. Технические науки. – 2008. – № 1. – С. 59–63.

4. Mohd. Ehmer Khan Importance of Software Testing in Software Development Life Cycle / Mohd. Ehmer Khan, Farmeena Khan // IJCSI International Journal of Computer Science Issues, Vol. 11, Issue 2, No 2, March 2014.

5. Кудрявцева Е. Ю. Автоматизированное тестирование веб-интерфейсов / Е. Ю. Кудрявцева // Горный информационно-аналитический бюллетень : научно-технический журнал. – 2014. – № S. – С. 354–356.

6. Яковенко П. Н. Инфраструктура тестирования веб-сервисов на базе технологии TTCN-3 и платформы. Net / П. Н. Яковенко, А. В. Сапожников // Труды ИСП РАН. – 2009. – № 17. – С. 63–74.

7. Силаков Д. В. Автоматизация тестирования Web-приложений, основанных на скриптовых языках / Д. В. Силаков // Труды ИСП РАН. – 2008. – № 2. – С. 159–178.

8. James T. The Docker Book: Containerization is the new virtualization. Amazon Digital Services LLC. – 2018. – 387 p.

Стаття надійшла до редакції 11.04.2018 р.