МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВАДИМА ГЕТЬМАНА

Навчально-науковий інститут
«Інститут інформаційних технологій в економіці»

Кафедра інформаційних систем в економіці

ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА
«Інформаційні управляючі системи і технології»

галузь знань 12 Інформаційні технології

Спеціальність 122 Комп'ютерні науки

Форма навчання: заочна

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:
«Інформаційна система управління процесами тестування та виявлення дефектів програмного забезпечення»

Здобувача **Кукси Катерини Миколаївни**

Науковий керівник: д.е.н., професор Мозгаллі О. П.

_____
(підпис)

Робота допущена до захисту перед екзаменаційною комісією з атестації здобувачів вищої освіти (ЕК)

Завідувач кафедри: к.е.н., доцент Тішков Б. О.

_____
(підпис)

Київ 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВАДИМА ГЕТЬМАНА
Навчально-науковий інститут «Інститут інформаційних технологій в економіці»
Кафедра інформаційних систем в економіці

ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА
«Інформаційні управляючі системи і технології»

галузь знань 12 Інформаційні технології
Спеціальність 122 Комп'ютерні науки

ПОГОДЖЕНО:
Керівник проєктної групи (гарант)
освітньо-професійної програми

_____Устенко С.В.
«23» січня 2024 р.

ЗАТВЕРДЖУЮ:
Завідувач кафедри інформаційних
систем в економіці

_____ Тішков Б.О.
«23» січня 2024 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

здобувача вищої освіти Кукси Катерини Миколаївни

заочної форми навчання

на підготовку кваліфікаційної магістерської роботи
на тему: *«Інформаційна система управління процесами тестування та виявлення дефектів програмного забезпечення»*

Тему затверджено наказом ректора Університету від «23» січня 2024р. №127-ст

Кваліфікаційна магістерська робота виконується на матеріалах "Telefon chi org" MCHJ

План кваліфікаційної магістерської роботи

Розділ І: Дослідження та аналіз інформаційних систем управлінням процесами тестуванням та виявленням дефектів програмного забезпечення.
Розділ ІІ: Характеристика системи та методів управлінням процесами тестуванням та виявленням дефектів програмного забезпечення.
Розділ ІІІ: Розроблення проєктних рішень для інформаційної системи управлінням тестуванням та виявленням дефектів програмного забезпечення.
Об'єкт дослідження: процеси тестування та виявлення дефектів програмного забезпечення, а також інформаційні системи та технології, які використовуються для управління цими процесами.
Предмет дослідження: моделі, методи, підходи та проєктні рішення, які використовуються при створенні інформаційних систем управління процесами тестування та виявлення дефектів програмного забезпечення, зокрема в частині використання інтелектуального аналізу даних та автоматизації тестування.

Мета кваліфікаційної магістерської роботи: Мета дослідження полягає в розробці теоретичних засад та практичних рішень для створення інформаційної системи управління процесами тестування та виявлення дефектів програмного забезпечення, яка б дозволила підвищити ефективність та якість цих процесів за рахунок використання сучасних методів та технологій інтелектуального аналізу даних, автоматизації тестування та оптимізації управління тестуванням.

Конкретні завдання, які здобувач повинен виконати для досягнення поставленої мети:

У розділі І провести аналіз сучасного стану та тенденцій розвитку інформаційних систем управління процесами тестування та виявлення дефектів програмного забезпечення, виявити їх ключові характеристики, переваги та недоліки. Дослідити теоретичні засади та сучасні підходи до створення інформаційних систем управління процесами тестування програмного забезпечення, зокрема в частині використання методів інтелектуального аналізу даних та автоматизації тестування.

У розділі ІІ розробити архітектуру та структурну модель інформаційної системи управління процесами тестування та виявлення дефектів програмного забезпечення, яка б враховувала сучасні вимоги та тенденції в цій галузі. Дослідити та обґрунтувати вибір методів та моделей інтелектуального аналізу даних для виявлення прихованих закономірностей та залежностей в даних про дефекти програмного забезпечення, а також для прогнозування потенційних проблемних місць в коді.

У розділі ІІІ розробити проєктні рішення щодо реалізації інформаційної системи управління процесами тестування, зокрема в частині проєктування бази даних, засобів інтелектуального аналізу даних, програмних модулів та компонентів системи, а також вибору необхідного технічного забезпечення. Реалізувати інформаційну систему управління процесами тестування та виявлення дефектів програмного забезпечення, перевірити її роботу на реальних даних, оцінити її ефективність та відповідність поставленим вимогам

Завдання підготував науковий керівник _____ О.П. Мозгаллі
(підпис)
«23» січня 2024 р.

Завдання одержав здобувач _____ К. М. Кукса
(підпис)
«23» січня 2024 р.

Реферат

Кваліфікаційна магістерська робота містить 68 сторінок, таблиці, рисунки, список використаних джерел з 40 найменувань, додатки.

«Інформаційні системи управління процесами тестування та виявлення дефектів програмного забезпечення»

Об'єктом дослідження кваліфікаційної магістерської роботи виступають процеси тестування та виявлення дефектів програмного забезпечення, а також інформаційні системи та технології, які використовуються для управління цими процесами.

Предметом дослідження є моделі, методи, підходи та проєктні рішення, які використовуються при створенні інформаційних систем управління процесами тестування та виявлення дефектів програмного забезпечення, зокрема в частині використання інтелектуального аналізу даних та автоматизації тестування.

Мета і завдання дослідження. Мета дослідження полягає в розробці теоретичних засад та практичних рішень для створення інформаційної системи управління процесами тестування та виявлення дефектів програмного забезпечення, яка б дозволила підвищити ефективність та якість цих процесів за рахунок використання сучасних методів та технологій інтелектуального аналізу даних, автоматизації тестування та оптимізації управління тестуванням.

Для досягнення поставленої мети в роботі вирішуються наступні завдання:

- Провести аналіз сучасного стану та тенденцій розвитку інформаційних систем управління процесами тестування та виявлення дефектів програмного забезпечення, виявити їх ключові характеристики, переваги та недоліки.
- Дослідити теоретичні засади та сучасні підходи до створення інформаційних систем управління процесами тестування програмного забезпечення, зокрема в частині використання методів інтелектуального аналізу даних та автоматизації тестування.
- Розробити архітектуру та структурну модель інформаційної системи управління процесами тестування та виявлення дефектів програмного забезпечення, яка б враховувала сучасні вимоги та тенденції в цій галузі.
- Дослідити та обґрунтувати вибір методів та моделей інтелектуального аналізу даних для виявлення прихованих закономірностей та залежностей в даних про дефекти програмного забезпечення, а також для прогнозування потенційних проблемних місць в коді.
- Розробити проєктні рішення щодо реалізації інформаційної системи управління процесами тестування, зокрема в частині проєктування бази даних, засобів інтелектуального аналізу даних, програмних модулів та компонентів системи, а також вибору необхідного технічного забезпечення.
- Реалізувати інформаційну систему управління процесами тестування та виявлення дефектів програмного забезпечення, перевірити її роботу на реальних даних, оцінити її ефективність та відповідність поставленим вимогам.

Теоретична, методична та практична значущість отриманих результатів. Під час дослідження було досліджено теоретичні сучасні підходи до створення інформаційних систем управління процесами тестування програмного забезпечення, результати дослідження полягають у створенні інформаційної системи, яка об’єднає в собі весь функціонал управління тестуванням в єдиній програмі, що є вигідним для компаній з точки зору оптимізації витрат та ефективного використання часу.

Рік виконання кваліфікаційної магістерської роботи – 2024.

Рік захисту роботи – 2024.

Ключові слова: контроль якості, програмне забезпечення, тестування, виявлення дефектів

Відгук
про кваліфікаційну магістерську роботу
здобувача навчально-наукового інституту
«Інститут інформаційних технологій в економіці»
освітньо-професійної програми «Інформаційні управляючі системи і технології»

Кукси Катерини Миколаївни

на тему ***«Інформаційна система управління процесами тестування та виявлення дефектів програмного забезпечення»***

1. Актуальність теми. Актуальність теми дослідження зумовлена стрімким розвитком інформаційних технологій та зростаючою роллю програмного забезпечення в сучасному світі, оскільки від надійності, безпечності та ефективності роботи програмних систем залежить не лише продуктивність та ефективність бізнес-процесів, але й довіра та лояльність клієнтів, репутація компанії на ринку. Проте розроблення якісного програмного забезпечення є складним та трудомістким процесом, який вимагає значних часових, людських та фінансових ресурсів, і невід'ємною частиною цього процесу є тестування програмного забезпечення та виявлення дефектів, оскільки на етапі тестування можна виявити та усунути більшість помилок, недоліків та вразливостей програмного продукту, забезпечивши таким чином його високу якість та відповідність вимогам замовника. Саме тому обрана тема кваліфікаційної магістерської роботи є актуальною та своєчасною.

2. Позитивні риси кваліфікаційної магістерської роботи. Розроблена автором інформаційна система управління тестуванням та виявленням дефектів програмного забезпечення дозволяє ефективно управляти процесами тестування, відстежувати дефекти, генерувати звіти та проводити аналітику даних. Система забезпечує підвищення якості програмного забезпечення, оптимізацію роботи команди тестування та прийняття обґрунтованих рішень на основі даних. Результати кваліфікаційної магістерської роботи можуть бути використані для подальшого вдосконалення процесів тестування та управління якістю програмного забезпечення в організаціях. Розроблена інформаційна система може слугувати основою для майбутніх досліджень та розширення функціональних можливостей у цій предметній області.

3. Наявність самостійних розробок автора. Представлена кваліфікаційна магістерська робота є самостійною розробкою автора, зокрема в роботі розроблено засоби інтелектуального аналізу даних, які дозволяють виявляти закономірності, тенденції та взаємозв'язки у даних про тестування та дефекти, а також допомагають

ухвалювати обґрунтовані рішення щодо планування тестування, розподілу ресурсів та вдосконалення процесів розробки програмного забезпечення. Також реалізовано веб-додаток інформаційної системи з використанням сучасних технологій та фреймворків, таких як PHP та Laravel, і розроблено різноманітні модулі та функції, які забезпечують ефективне управління проєктами, тестовими сценаріями, дефектами, генерацію звітів та аналітику даних.

4. Цінність теоретичних висновків та практичних рекомендацій. У ході виконання кваліфікаційної магістерської роботи Кукса К.М. проаналізувала наявні теоретичні підходи до створення інформаційних систем управління процесами тестування програмного забезпечення, на основі чого розробила проєкт інформаційної системи, яка поєднує весь функціонал управління тестуванням в єдиній програмі, що є вигідним для компаній з погляду оптимізації витрат та ефективного використання часу

5. Наявність недоліків. Представлене проєктне рішення потребує подальшого вдосконалення й розвитку для використання у промисловій експлуатації.

6. Загальна оцінка кваліфікаційної магістерської роботи та її допущення до захисту перед ЕК. Загалом, представлена кваліфікаційна магістерська робота відповідає встановленим вимогам методичних вказівок щодо структури, обсягу та змісту та **рекомендується до захисту** з позитивною оцінкою.

Науковий керівник: *професор кафедри інформаційних систем в економіці, професор, д.е.н.*

Мозгаллі О. П.

“10” травня 2024 р.

Рецензія

на кваліфікаційну магістерську роботу
здобувача вищої освіти
Кукси Катерини Миколаївни
На тему: «Інформаційна система управління процесами тестування та виявлення дефектів програмного забезпечення».

Актуальність теми кваліфікаційної магістерської роботи і доцільність її розроблення. Актуальність теми КМР очевидна у зв'язку з постійною потребою в покращенні процесів тестування програмного забезпечення. Розробка інформаційної системи є відповіддю на цю потребу та має потенціал значно покращити якість та ефективність розробки програм.

Якість проведеного дослідження. Робота детально проаналізована та систематизована інформацією про методи управління тестуванням та виявленням дефектів програмного забезпечення, використовуючи актуальні теоретичні підходи та практичні приклади. Такий підхід дозволяє нам впевнено стверджувати, що дослідження відповідає вимогам наукової обґрунтованості та методологічної правильності.

Позитивні риси кваліфікаційної магістерської роботи. Ця робота вражає своєю практичністю та чітким викладом, що робить його зрозумілим для всіх. Глибокий аналіз проблеми, має великий потенціал у реальному застосуванні.

Зауваження. Хоча робота має свої позитивні моменти, вона також містить деякі зауваження щодо недостатньої деталізації деяких аспектів, проте це зауваження не є вирішальним для загального позитивного враження від роботи.

Практична значимість висновків і рекомендацій. Результати дослідження мають великий вплив на покращення управління тестуванням та виявленням дефектів програмного забезпечення. Розроблена інформаційна система може автоматизувати процеси тестування та підвищити якість продуктів. Рекомендації, зроблені у цій роботі, будуть корисні для фахівців у цій галузі, допомагаючи оптимізувати управління тестуванням та підвищувати продуктивність розробки.

Рецензент:
"Telefon chi org" MCHJ, продакт-менеджер _____ Кім С. В.
Підпис засвідчую:
Начальник відділу тестування _____ Султанов Ю. Р.

ЗМІСТ

ВСТУП

Актуальність теми дослідження зумовлена стрімким розвитком інформаційних технологій та зростаючою роллю програмного забезпечення в сучасному світі. В умовах високої конкуренції та швидкого темпу життя якість програмного забезпечення стає одним із ключових факторів успіху будь-якого бізнесу чи організації. Адже саме від надійності, безпечності та ефективності роботи програмних систем залежить не лише продуктивність та ефективність бізнес-процесів, але й довіра та лояльність клієнтів, репутація компанії на ринку.

Однак, розробка якісного програмного забезпечення є складним та трудомістким процесом, який вимагає значних часових, людських та фінансових ресурсів. Невід'ємною частиною цього процесу є тестування програмного забезпечення та виявлення дефектів. Саме на етапі тестування можна виявити та усунути більшість помилок, недоліків та вразливостей програмного продукту, забезпечивши таким чином його високу якість та відповідність вимогам замовника[3, 4].

В той же час, процес тестування програмного забезпечення сам по собі є достатньо складним та трудомістким, особливо враховуючи зростаючі масштаби та складність сучасних програмних систем. Традиційні підходи до організації та проведення тестування, які базуються на ручній роботі тестувальників та використанні простих інструментів, вже не можуть забезпечити необхідний рівень якості та ефективності цього процесу. Адже, по-перше, ручне тестування є дуже повільним та не встигає за темпами розробки програмного забезпечення. По-друге, воно є дуже дорогим, оскільки вимагає залучення великої кількості висококваліфікованих спеціалістів. По-третє, ручне тестування не може охопити всі можливі сценарії використання програмного продукту та виявити всі потенційні дефекти[5].

Саме тому в сучасних умовах все більшої актуальності набуває питання автоматизації та оптимізації процесів тестування програмного забезпечення за допомогою спеціалізованих інформаційних систем управління. Такі системи дозволяють значно підвищити ефективність та якість тестування, зменшити часові

та фінансові витрати на цей процес, забезпечити більш повне покриття коду тестами та виявлення більшої кількості дефектів.

Сучасні інформаційні системи управління процесами тестування програмного забезпечення базуються на використанні передових інформаційних технологій та підходів, таких як інтелектуальний аналіз даних, машинне навчання, автоматизація тестування, хмарні обчислення тощо. Ці технології дозволяють не лише автоматизувати рутинні процеси тестування, але й виявляти приховані закономірності та залежності в даних про дефекти, прогнозувати потенційні проблеми в роботі програмного забезпечення, оптимізувати процес тестування на основі аналізу ризиків та пріоритетів[6].

Таким чином, розробка та впровадження інформаційних систем управління процесами тестування та виявлення дефектів програмного забезпечення є надзвичайно актуальним та перспективним напрямком досліджень, який має велике теоретичне та практичне значення. З одного боку, такі системи дозволяють підвищити якість та надійність програмного забезпечення, зменшити ризики та витрати, пов'язані з його розробкою та експлуатацією. З іншого боку, вони відкривають нові можливості для досліджень в галузі інформаційних технологій, зокрема в сферах інтелектуального аналізу даних, машинного навчання, автоматизації процесів тощо.

Незважаючи на значну кількість досліджень та розробок в цій галузі, все ще існує багато невирішених проблем та викликів, пов'язаних зі створенням ефективних та надійних інформаційних систем управління процесами тестування програмного забезпечення. Зокрема, актуальними залишаються питання вибору оптимальних методів та моделей інтелектуального аналізу даних для виявлення дефектів, розробки гнучких та адаптивних архітектур таких систем, інтеграції різних інструментів та середовищ тестування, забезпечення безпеки та конфіденційності даних в процесі тестування тощо.

Все це зумовлює необхідність проведення подальших теоретичних та прикладних досліджень в цій галузі, розробки нових підходів, методів та інструментів для створення ефективних інформаційних систем управління

процесами тестування та виявлення дефектів програмного забезпечення. Саме на вирішення цих завдань і спрямована дана магістерська робота.

Мета дослідження полягає в розробці теоретичних засад та практичних рішень для створення інформаційної системи управління процесами тестування та виявлення дефектів програмного забезпечення, яка б дозволила підвищити ефективність та якість цих процесів за рахунок використання сучасних методів та технологій інтелектуального аналізу даних, автоматизації тестування та оптимізації управління тестуванням.

Для досягнення поставленої мети в роботі вирішуються наступні завдання:

- Провести аналіз сучасного стану та тенденцій розвитку інформаційних систем управління процесами тестування та виявлення дефектів програмного забезпечення, виявити їх ключові характеристики, переваги та недоліки.
- Дослідити теоретичні засади та сучасні підходи до створення інформаційних систем управління процесами тестування програмного забезпечення, зокрема в частині використання методів інтелектуального аналізу даних та автоматизації тестування.
- Розробити архітектуру та структурну модель інформаційної системи управління процесами тестування та виявлення дефектів програмного забезпечення, яка б враховувала сучасні вимоги та тенденції в цій галузі.
- Дослідити та обґрунтувати вибір методів та моделей інтелектуального аналізу даних для виявлення прихованих закономірностей та залежностей в даних про дефекти програмного забезпечення, а також для прогнозування потенційних проблемних місць в коді.
- Розробити проєктні рішення щодо реалізації інформаційної системи управління процесами тестування, зокрема в частині проєктування бази даних, засобів інтелектуального аналізу даних, програмних модулів та компонентів системи, а також вибору необхідного технічного забезпечення.
- Реалізувати інформаційну систему управління процесами тестування та виявлення дефектів програмного забезпечення, перевірити її роботу на реальних даних, оцінити її ефективність та відповідність поставленим вимогам.

Об'єктом дослідження в роботі виступають процеси тестування та виявлення дефектів програмного забезпечення, а також інформаційні системи та технології, які використовуються для управління цими процесами.

Предметом дослідження є моделі, методи, підходи та проєктні рішення, які використовуються при створенні інформаційних систем управління процесами тестування та виявлення дефектів програмного забезпечення, зокрема в частині використання інтелектуального аналізу даних та автоматизації тестування.

Методи дослідження, які використовуються в роботі, включають: системний аналіз, методи моделювання бізнес-процесів та інформаційних систем, методи проєктування баз даних та програмного забезпечення, методи математичної статистики та інтелектуального аналізу даних, зокрема методи класифікації, кластеризації, асоціативних правил, регресійного аналізу, нейронних мереж тощо.

Наукова новизна одержаних результатів полягає в розробці нових моделей, методів та проєктних рішень для створення інформаційних систем управління процесами тестування та виявлення дефектів програмного забезпечення на основі використання сучасних підходів до інтелектуального аналізу даних та автоматизації тестування. Зокрема, в роботі пропонується оригінальна архітектура такої системи, яка базується на мікросервісному підході та використанні хмарних технологій. Також розроблено нові методи виявлення прихованих закономірностей та залежностей в даних про дефекти програмного забезпечення на основі використання ансамблів моделей машинного навчання та нейронних мереж глибокого навчання.

Практичне значення одержаних результатів полягає в тому, що розроблена інформаційна система управління процесами тестування та виявлення дефектів програмного забезпечення може бути використана в реальних проєктах з розробки програмних продуктів для підвищення ефективності та якості процесів тестування, зменшення ризиків та витрат, пов'язаних з виявленням та усуненням дефектів. Запропоновані в роботі моделі та методи інтелектуального аналізу даних можуть бути використані для автоматизації процесів виявлення дефектів, прогнозування потенційних проблемних місць в коді, оптимізації процесів тестування на основі аналізу ризиків та пріоритетів. Все це дозволяє значно підвищити якість та

надійність програмного забезпечення, зменшити часові та фінансові витрати на його розробку та супровід.

Результати кваліфікаційної магістерської роботи доповідались та обговорювались на IX Міжнародній науково-практичній конференції «Інноваційне підприємництво: стан та перспективи розвитку» та V Міжнародній науково-практичній конференції молодих вчених, аспірантів і студентів «Сучасні інформаційні технології та системи в управлінні».

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ТА АНАЛІЗ ІНФОРМАЦІЙНИХ СИСТЕМ УПРАВЛІННЯ ПРОЦЕСАМИ ТЕСТУВАННЯ ТА ВИЯВЛЕННЯ ДЕФЕКТІВ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Огляд та аналіз існуючих інформаційних систем у предметній області

У сучасному світі інформаційні технології відіграють ключову роль у забезпеченні ефективності та якості процесів розробки програмного забезпечення. Одним із найважливіших етапів цього процесу є тестування та виявлення дефектів, адже саме від цього залежить надійність, безпечність та продуктивність кінцевого продукту.

Для управління процесами тестування та виявлення дефектів програмного забезпечення використовуються спеціалізовані інформаційні системи, які дозволяють автоматизувати та оптимізувати ці процеси, зменшити витрати часу та ресурсів, підвищити якість та ефективність тестування[1, 2].

На ринку представлено достатньо широкий спектр інформаційних систем управління процесами тестування програмного забезпечення, які відрізняються своїми функціональними можливостями, архітектурою, використовуваними технологіями та підходами. Розглянемо деякі з найбільш популярних та ефективних рішень в цій галузі.

Однією з найвідоміших інформаційних систем управління процесами тестування є HP Quality Center (зараз відома як Micro Focus ALM Quality Center). Ця система є комплексним рішенням для управління якістю програмного забезпечення на всіх етапах його життєвого циклу, від планування та аналізу вимог до тестування та випуску продукту.

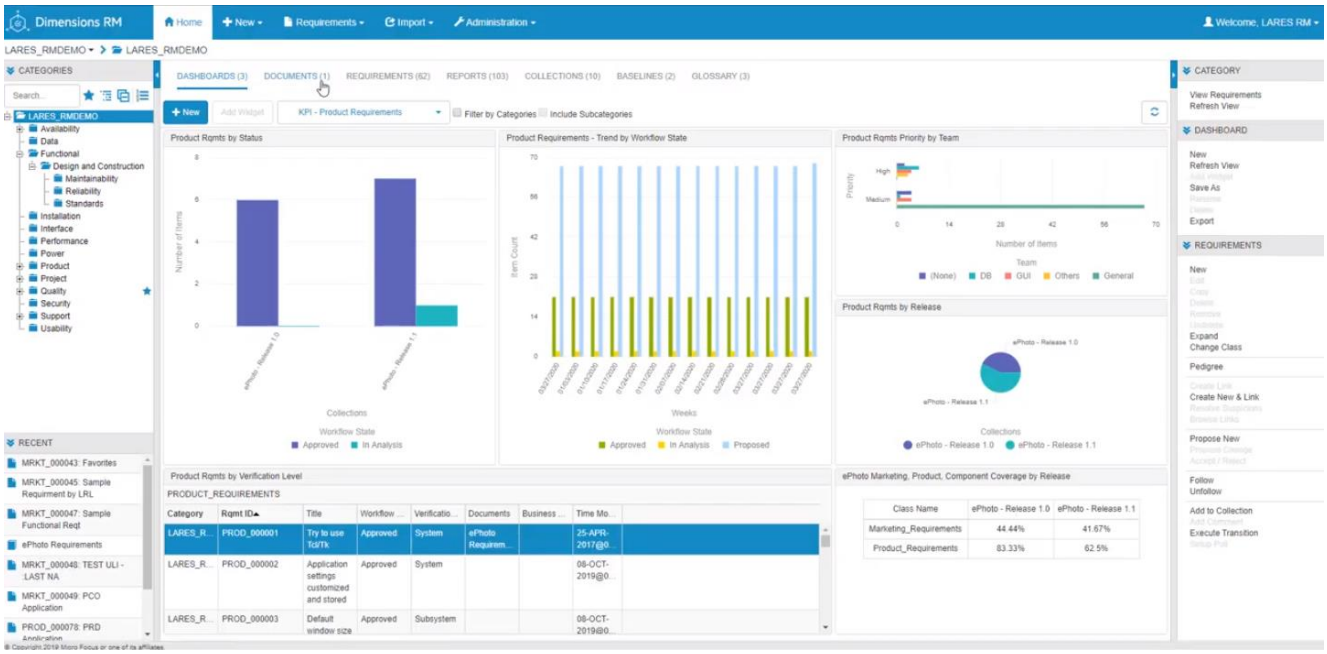

Рисунок 1.1 - Логотип компанії Micro Focus

HP Quality Center забезпечує єдину платформу для співпраці всіх учасників процесу розробки програмного забезпечення, включаючи менеджерів проєктів, аналітиків, розробників, тестувальників та інших зацікавлених сторін. Система дозволяє управляти вимогами та тестовими кейсами, планувати та відслідковувати виконання тестів, збирати та аналізувати дані про дефекти, генерувати звіти та метрики якості.

Серед ключових переваг HP Quality Center можна виділити:

- Комплексне управління вимогами та тестовими кейсами на всіх рівнях тестування (модульне, інтеграційне, системне, приймальне)
- Гнучке планування та відслідковування виконання тестів з використанням різних методологій (Waterfall, Agile, Hybrid)
- Потужні засоби для збору, класифікації та аналізу даних про дефекти з можливістю інтеграції з системами відслідковування помилок (bug tracking)
- Автоматизація тестування з використанням інструментів HP Unified Functional Testing (UFT) та HP LoadRunner
- Генерація різноманітних звітів та метрик якості для оцінки стану проєкту та прийняття управлінських рішень

- Можливість інтеграції з іншими системами управління життєвим циклом програмного забезпечення (ALM) та інструментами безперервної інтеграції (CI)

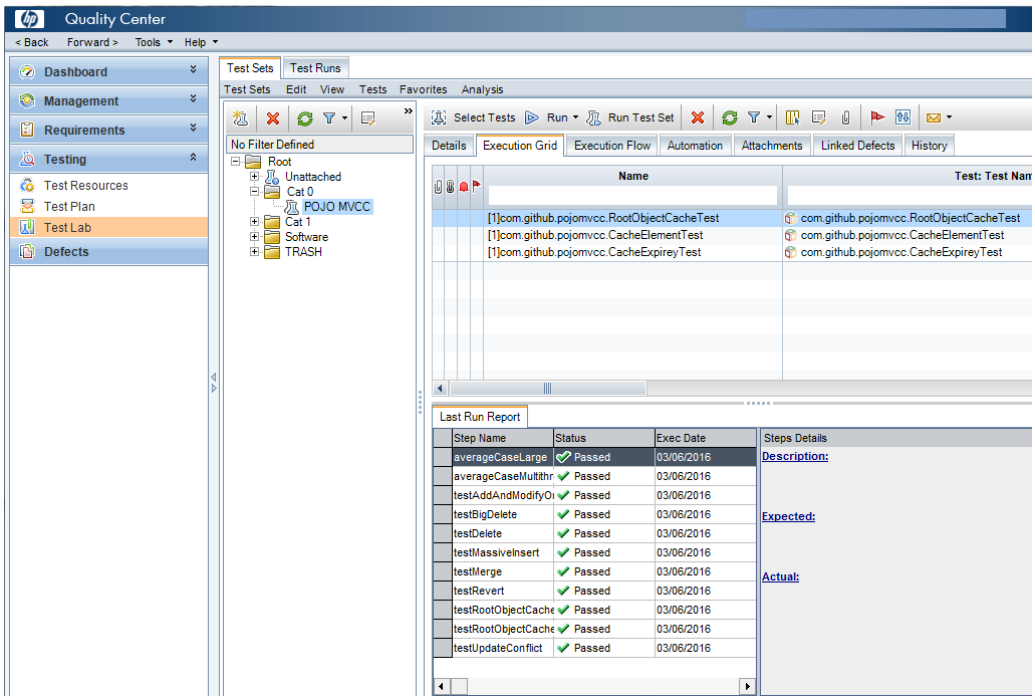

Рисунок 1.2 - вигляд інтерфейсу HP Quality Center

Іншим популярним рішенням для управління процесами тестування є IBM Rational Quality Manager. Ця система є частиною платформи IBM Rational CLM (Collaborative Lifecycle Management) і забезпечує комплексне управління якістю програмного забезпечення на всіх етапах його розробки.

IBM Rational Quality Manager дозволяє планувати та відслідковувати тестування, управляти тестовими активами (тестові кейси, скрипти, дані), збирати та аналізувати результати тестів, а також координувати роботу всіх учасників процесу тестування. Система підтримує різні типи та рівні тестування, включаючи ручне та автоматизоване тестування, функціональне та нефункціональне тестування, модульне та інтеграційне тестування тощо.

Серед ключових особливостей IBM Rational Quality Manager можна виділити:

- Управління тестовими активами з використанням бібліотек та шаблонів тестових кейсів

- Планування та відслідковування тестових циклів з використанням різних методологій (Waterfall, Agile, DevOps)
- Збір та аналіз результатів тестів з можливістю інтеграції з системами відслідковування помилок (Jira, Bugzilla, Rational Team Concert)
- Автоматизація тестування з використанням інструментів Rational Functional Tester та Rational Performance Tester
- Можливість створення власних метрик та звітів для оцінки якості та прогресу тестування
- Інтеграція з іншими інструментами платформи IBM Rational CLM (Rational Team Concert для управління вимогами та завданнями, Rational DOORS Next Generation для управління вимогами)

Ще одним потужним інструментом для управління процесами тестування є Microsoft Test Manager (MTM). Ця система є частиною платформи Microsoft Visual Studio і забезпечує планування, виконання та відслідковування тестування для проєктів, що розробляються на платформі Microsoft.

Microsoft Test Manager дозволяє створювати та управляти тестовими планами, тестовими кейсами та тестовими сценаріями, виконувати ручне та дослідницьке тестування, збирати та аналізувати результати тестів, а також координувати роботу тестувальників. Система інтегрується з іншими інструментами платформи Visual Studio, такими як Team Foundation Server (TFS) для управління вимогами та завданнями, а також з інструментами автоматизації тестування, такими як Coded UI Tests та Selenium.

Серед ключових переваг Microsoft Test Manager можна виділити:

- Інтеграція з платформою Visual Studio та іншими інструментами Microsoft для розробки та управління життєвим циклом програмного забезпечення
- Підтримка різних типів та методів тестування, включаючи дослідницьке тестування, тестування на основі вимог, регресійне тестування тощо
- Можливість створення та виконання тестових сценаріїв з використанням записаних дій користувача (action recordings)
- Автоматизація тестування з використанням Coded UI Tests та інтеграція з популярними фреймворками автоматизації, такими як Selenium

- Аналіз впливу змін коду на тестові кейси з використанням інструментів статичного аналізу коду
- Генерація різноманітних звітів та метрик для оцінки стану та якості тестування

Окрім розглянутих вище систем, на ринку представлено ще багато інших інструментів для управління процесами тестування програмного забезпечення, серед яких можна виділити такі як:

- TestRail - веб-орієнтована система для управління тестовими кейсами, планування та відслідковування тестування, генерації звітів. Підтримує інтеграцію з популярними системами відслідковування помилок (Jira, Bugzilla, Redmine) та інструментами автоматизації тестування.
- qTest - хмарна платформа для управління тестуванням, яка забезпечує планування та відслідковування тестових активностей, управління тестовими кейсами та дефектами, автоматизацію тестування, а також аналітику та звітність. Підтримує інтеграцію з різноманітними ALM-системами та інструментами розробки.
- Zephyr - сімейство продуктів для управління тестуванням, яке включає в себе такі інструменти як Zephyr for Jira (додаток для управління тестуванням в середовищі Jira), Zephyr Scale (хмарна платформа для масштабованого управління тестуванням) та Zephyr Squad (інструмент для управління тестуванням в Agile-командах).
- TestLink - відкрита веб-орієнтована система управління тестуванням, яка дозволяє планувати та відслідковувати тестові активності, управляти тестовими кейсами та тестовими планами, а також генерувати звіти та метрики. Система є безкоштовною та має відкритий вихідний код.

Підсумовуючи огляд існуючих інформаційних систем управління процесами тестування програмного забезпечення, можна зробити висновок, що на ринку представлено достатньо широкий вибір рішень, які відрізняються своїми функціональними можливостями, архітектурою, використовуваними технологіями та підходами.

При виборі конкретної системи для використання в своїх проєктах необхідно враховувати такі фактори, як масштаб та складність проєкту, використовувані методології та практики тестування, наявні ресурси та бюджет, а також сумісність з існуючими інструментами та платформами розробки.

Однак, незважаючи на достатньо широкий вибір існуючих рішень, все ще існує потреба в розробці нових та вдосконаленні існуючих інформаційних систем управління процесами тестування, які б врахували сучасні тенденції та виклики в галузі розробки програмного забезпечення, такі як:

- Зростання складності та обсягів програмних систем, що вимагає більш ефективних та масштабованих підходів до тестування
- Перехід до гнучких (Agile) методологій розробки, що вимагає більш тісної інтеграції процесів тестування з процесами розробки та управління вимогами
- Використання штучного інтелекту та машинного навчання для автоматизації та оптимізації процесів тестування, виявлення та прогнозування дефектів
- Необхідність забезпечення безпеки та конфіденційності даних в процесі тестування, особливо в контексті розробки програмного забезпечення для критичних галузей (фінанси, медицина, енергетика тощо)
- Підвищення вимог до швидкості та якості випуску програмних продуктів в умовах безперервної інтеграції та доставки (CI/CD)

Продовжуючи огляд існуючих інформаційних систем управління процесами тестування, варто також звернути увагу на деякі інші рішення, які можуть бути корисними для різних типів проєктів та команд розробки.

Однією з таких систем є PractiTest - хмарна платформа для управління процесами тестування, яка орієнтована на гнучкі (Agile) методології розробки. PractiTest дозволяє створювати та управляти тестовими кейсами, планувати та відслідковувати тестові активності, збирати та аналізувати результати тестів, а також забезпечувати комунікацію та співпрацю між членами команди.

Серед ключових переваг PractiTest можна виділити:

- Підтримка різних типів тестування, включаючи дослідницьке, регресійне, функціональне та нефункціональне тестування

- Можливість інтеграції з популярними інструментами розробки, такими як Jira, Jenkins, Selenium, Slack тощо
- Гнучке налаштування процесів тестування під потреби конкретної команди або проєкту
- Потужні засоби для аналітики та звітності, включаючи кастомні дашборди та метрики
- Високий рівень безпеки та конфіденційності даних завдяки використанню хмарної інфраструктури

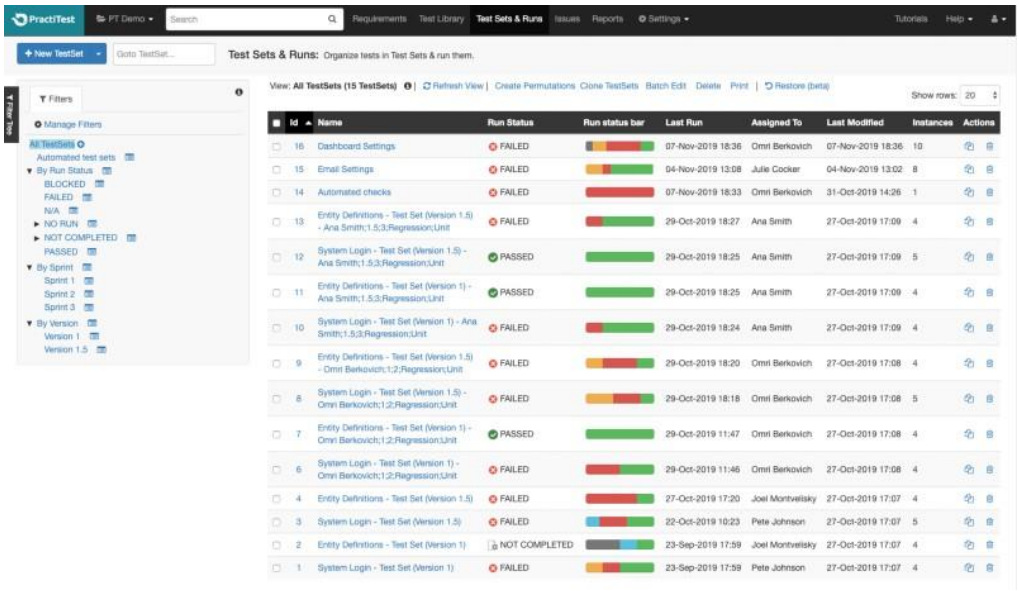

Рисунок 1.3 - Приклад інтерфейсу PractiTest

Ще однією системою, яка заслуговує на увагу, є Testuff - веб-орієнтована платформа для управління тестуванням, яка підтримує як ручне, так і автоматизоване тестування. Testuff дозволяє створювати та управляти тестовими кейсами, планувати та відслідковувати тестові активності, збирати та аналізувати результати тестів, а також генерувати звіти та метрики.

Серед ключових особливостей Testuff можна виділити:

- Інтуїтивно зрозумілий та зручний користувацький інтерфейс
- Підтримка різних типів тестування, включаючи функціональне, регресійне, дослідницьке та приймальне тестування

- Можливість інтеграції з популярними системами відслідковування помилок (Jira, Bugzilla, Redmine) та інструментами автоматизації тестування (Selenium, Appium, Jenkins)
- Гнучке управління правами доступу та ролями користувачів
- Потужні засоби для аналітики та звітності, включаючи кастомні метрики та графіки

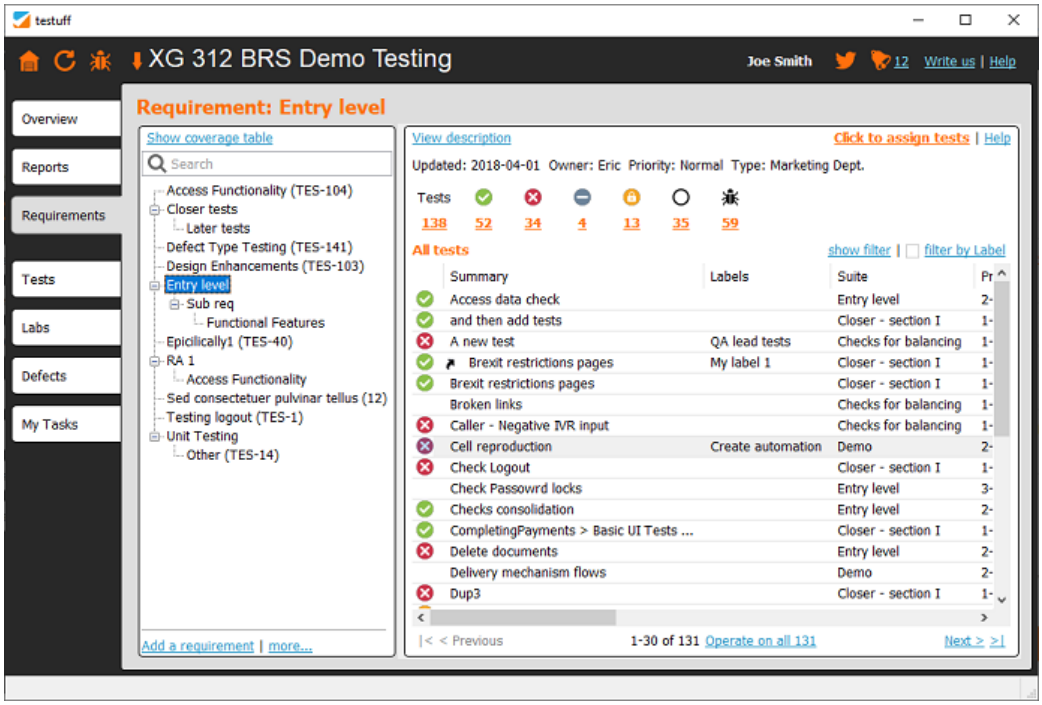

Рисунок 1.4 - Дашборд тестування в Testuff

Окрім розглянутих вище систем, також варто згадати про такі рішення, як:

- TestCaselab - веб-орієнтована система для управління тестовими кейсами та тестовими сценаріями, яка підтримує як ручне, так і автоматизоване тестування. TestCaselab дозволяє створювати та управляти тестовими кейсами, планувати та відслідковувати тестові активності, а також генерувати звіти та метрики.
- QACoverage - веб-орієнтована система для управління процесами тестування, яка фокусується на забезпеченні якості та покритті вимог тестами. QACoverage дозволяє управляти вимогами та тестовими кейсами, відслідковувати прогрес та статус тестування, а також генерувати звіти та метрики якості.

- Kualitee - хмарна платформа для управління процесами тестування, яка підтримує як ручне, так і автоматизоване тестування. Kualitee дозволяє створювати та управляти тестовими кейсами, планувати та відслідковувати тестові активності, збирати та аналізувати результати тестів, а також забезпечувати комунікацію та співпрацю між членами команди.

Для кращого розуміння відмінностей та переваг різних інформаційних систем управління процесами тестування, наведемо порівняльну таблицю деяких з розглянутих вище рішень:

Таблиця 1.1 - Порівняльна таблиця інформаційних систем управління процесами тестування

Система	**Ключові особливості**	**Переваги**	**Недоліки**
HP Quality Center	– Комплексне управління вимогами та тестовими кейсами – Потужні засоби для аналізу дефектів – Автоматизація тестування з HP UFT та HP LoadRunner – Інтеграція з ALM-системами	– Широкі функціональні можливості – Потужна аналітика та звітність – Масштабованість та гнучкість – Інтеграція з іншими інструментами HP	– Висока вартість ліцензій – Складність налаштування та конфігурації – Необхідність спеціального навчання для користувачів
IBM Rational Quality Manager	– Управління тестовими активами та тестовими циклами – Підтримка різних типів та рівнів тестування	– Комплексне рішення для управління якістю – Потужні засоби для планування та відслідковування тестування	– Висока вартість ліцензій – Складність розгортання та конфігурації

Продовження таблиці 1.1

	– Автоматизація тестування з RFT та RPT – Інтеграція з платформою IBM Rational CLM	– Інтеграція з іншими інструментами IBM Rational	– Обмежена підтримка інструментів сторонніх вендорів
Microsoft Test Manager	– Інтеграція з платформою Visual Studio – Підтримка різних типів та методів тестування – Автоматизація тестування з Coded UI Tests – Аналіз впливу змін коду на тестові кейси	– Тісна інтеграція з інструментами Microsoft – Зручність використання для розробників та тестувальників – Підтримка дослідницького тестування	– Обмежена підтримка інструментів та платформ сторонніх вендорів – Необхідність використання інших інструментів Microsoft (TFS)
PractiTest	– Орієнтація на гнучкі методології розробки – Підтримка різних типів тестування – Інтеграція з популярними інструментами розробки	– Простота використання та налаштування – Гнучкість та адаптивність до процесів команди – Хороша інтеграція з Jira та іншими інструментами	– Обмежені можливості для автоматизації тестування – Відсутність вбудованих інструментів для управління вимогами

Продовження таблиці 1.1

Testuff	– Підтримка ручного та автоматизованого тестування – Інтеграція з системами відслідковування помилок та інструментами автоматизації – Гнучке управління правами доступу	– Зручний та інтуїтивно зрозумілий інтерфейс – Хороша інтеграція з популярними інструментами – Потужні засоби для аналітики та звітності	– Обмежені можливості для масштабування та налаштування процесів – Відсутність вбудованих інструментів для управління вимогами

Підсумовуючи огляд та аналіз існуючих інформаційних систем управління процесами тестування, можна зробити висновок, що вибір конкретного рішення залежить від специфічних потреб та вимог кожного окремого проєкту або організації. При цьому важливо враховувати такі фактори, як масштаб та складність проєкту, використовувані методології та практики розробки, наявні ресурси та бюджет, а також сумісність з існуючими інструментами та платформами.

Незважаючи на достатньо широкий вибір доступних рішень, все ще існує потреба в подальшому розвитку та вдосконаленні інформаційних систем управління процесами тестування, які б враховували сучасні тенденції та виклики в галузі розробки програмного забезпечення. Зокрема, актуальними напрямками досліджень та розробок в цій сфері є використання штучного інтелекту та машинного навчання для автоматизації та оптимізації процесів тестування, забезпечення безпеки та конфіденційності даних, а також підвищення швидкості та якості випуску програмних продуктів в умовах безперервної інтеграції та доставки.

Саме на вирішення цих та інших викликів і спрямовані сучасні дослідження та розробки в галузі інформаційних систем управління процесами тестування

програмного забезпечення, які будуть більш детально розглянуті в наступних розділах даної роботи.

1.2 Огляд підходів і технологій для створення інформаційної системи управління процесами тестування та виявлення дефектів програмного забезпечення

Створення ефективної та надійної інформаційної системи управління процесами тестування та виявлення дефектів програмного забезпечення вимагає використання сучасних підходів, методологій та технологій розробки. В даному підрозділі ми розглянемо основні з них, а також проаналізуємо їх переваги, недоліки та особливості застосування в контексті предметної області.

Перш за все, варто зазначити, що існує кілька ключових підходів до розробки інформаційних систем, серед яких можна виділити:

1. Каскадний підхід (Waterfall) - це традиційний підхід до розробки програмного забезпечення, який передбачає послідовне виконання етапів аналізу вимог, проєктування, розробки, тестування та впровадження. Кожен етап починається лише після повного завершення попереднього, і зміни вимог або дизайну на пізніх етапах проєкту є складними та затратними. Цей підхід є ефективним для проєктів з чітко визначеними та стабільними вимогами, але не підходить для динамічних та мінливих середовищ.
2. Ітеративний підхід - це підхід, при якому розробка системи відбувається в кілька ітерацій, кожна з яких включає в себе всі етапи розробки (аналіз, проєктування, розробку, тестування) для певної частини функціональності. Після завершення кожної ітерації отримується працююча версія продукту з обмеженою функціональністю, яка може бути продемонстрована замовнику та використана для отримання зворотного зв'язку. Цей підхід дозволяє швидше отримувати результати та адаптуватися до змін вимог, але вимагає більш ретельного планування та управління проєктом.

3. Гнучкий підхід (Agile) - це сімейство ітеративних методологій розробки, які орієнтовані на швидку та адаптивну розробку програмного забезпечення в умовах мінливих вимог та високої невизначеності. Agile-методології, такі як Scrum, Kanban, XP тощо, передбачають тісну співпрацю між замовником та командою розробки, регулярні ітерації (спринти) тривалістю 2-4 тижні, постійне покращення процесів та адаптацію до змін. Гнучкий підхід є ефективним для проєктів з динамічними вимогами та високим рівнем невизначеності, але вимагає високої кваліфікації та самоорганізації команди розробки.
4. DevOps - це підхід, який орієнтований на інтеграцію процесів розробки (Development) та експлуатації (Operations) програмного забезпечення з метою підвищення швидкості та якості випуску продуктів. DevOps передбачає автоматизацію процесів збірки, тестування та розгортання програмного забезпечення, використання практик безперервної інтеграції та доставки (CI/CD), а також тісну співпрацю між командами розробки, тестування та експлуатації. Цей підхід дозволяє значно зменшити час та витрати на випуск нових версій продукту, але вимагає значних інвестицій в інфраструктуру та навчання персоналу[35].

Окрім вибору загального підходу до розробки, важливим аспектом створення інформаційної системи управління процесами тестування є вибір конкретних технологій та інструментів. Розглянемо деякі з ключових технологій, які можуть бути використані для розробки такої системи:

1. Мови програмування - для розробки серверної частини системи можуть бути використані такі мови, як Java, C#, Python, Node.js тощо, в залежності від наявних ресурсів та кваліфікації команди розробки. Для розробки клієнтської частини (інтерфейсу користувача) можуть використовуватися такі технології, як HTML, CSS, JavaScript, а також фреймворки та бібліотеки, такі як React, Angular, Vue.js тощо.
2. Бази даних - для зберігання та управління даними системи можуть використовуватися як реляційні бази даних (наприклад, MySQL, PostgreSQL, Oracle), так і нереляційні (наприклад, MongoDB, Cassandra, Redis). Вибір

конкретної бази даних залежить від структури та обсягу даних, а також від вимог до продуктивності та масштабованості системи.

3. Фреймворки та бібліотеки - для прискорення та спрощення розробки системи можуть використовуватися різноманітні фреймворки та бібліотеки, такі як Spring Framework, Hibernate, JUnit для Java, .NET Framework, Entity Framework, NUnit для C#, Django, Flask, Pytest для Python тощо[24]. Ці інструменти надають готові рішення для типових задач розробки, таких як робота з базами даних, створення веб-сервісів, тестування коду тощо.
4. Інструменти автоматизації тестування - для автоматизації процесів тестування програмного забезпечення можуть використовуватися такі інструменти, як Selenium, Appium, Cucumber, JMeter тощо. Ці інструменти дозволяють створювати та виконувати автоматизовані тести для різних типів програмного забезпечення (веб-додатки, мобільні додатки, API тощо), а також генерувати звіти та метрики якості.
5. Інструменти безперервної інтеграції та доставки - для автоматизації процесів збірки, тестування та розгортання програмного забезпечення можуть використовуватися такі інструменти, як Jenkins, GitLab CI/CD, Travis CI, CircleCI тощо. Ці інструменти дозволяють налаштувати безперервну інтеграцію коду, автоматичне виконання тестів та розгортання додатків на різних середовищах (тестове, проміжне, продуктивне).
6. Хмарні платформи - для розгортання та масштабування інформаційної системи можуть використовуватися хмарні платформи, такі як Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform тощо. Ці платформи надають готову інфраструктуру та сервіси для розробки, тестування та експлуатації програмного забезпечення, а також забезпечують високу доступність, надійність та безпеку системи.

Окрім вибору конкретних технологій та інструментів, важливим аспектом створення інформаційної системи управління процесами тестування є також вибір відповідної архітектури та дизайну системи. Існує кілька поширених архітектурних стилів та шаблонів проєктування, які можуть бути використані для розробки такої системи, зокрема:

1. Багатошарова архітектура (N-Tier) - це архітектурний стиль, який передбачає розділення системи на кілька шарів (наприклад, шар представлення, шар бізнес-логіки, шар доступу до даних), кожен з яких відповідає за певний аспект функціональності системи. Цей підхід дозволяє розділити відповідальність між різними частинами системи, підвищити модульність та можливість повторного використання коду[23, 34].
2. Сервіс-орієнтована архітектура (SOA) - це архітектурний стиль, який передбачає розбиття системи на окремі сервіси, кожен з яких виконує певну бізнес-функцію та може бути використаний незалежно від інших сервісів. Сервіси взаємодіють між собою через стандартизовані інтерфейси (наприклад, веб-сервіси) та протоколи (наприклад, HTTP, SOAP, REST). Цей підхід дозволяє підвищити гнучкість та масштабованість системи, а також спростити інтеграцію з іншими системами.
3. Мікросервісна архітектура - це архітектурний стиль, який є подальшим розвитком сервіс-орієнтованої архітектури та передбачає розбиття системи на ще більш дрібні та незалежні сервіси (мікросервіси), кожен з яких відповідає за певну бізнес-функцію та може бути розроблений, протестований та розгорнутий окремо від інших сервісів. Мікросервіси взаємодіють між собою через прості та легкі протоколи (наприклад, HTTP, JSON) та можуть бути розгорнуті в контейнерах (наприклад, Docker) для забезпечення портативності та масштабованості.
4. Шаблони проєктування - це типові рішення для часто виникаючих проблем проєктування програмного забезпечення, які можуть бути використані для розробки окремих компонентів та модулів системи. Прикладами шаблонів проєктування, які можуть бути використані при розробці інформаційної системи управління процесами тестування, є:

- Шаблон Репозиторій (Repository) - для забезпечення доступу до даних та відокремлення бізнес-логіки від деталей зберігання даних.
- Шаблон Фабрика (Factory) - для створення об'єктів з різними конфігураціями або реалізаціями, наприклад, об'єктів тестових кейсів або дефектів.

- Шаблон Стратегія (Strategy) - для реалізації різних алгоритмів або підходів до вирішення певної задачі, наприклад, різних стратегій генерації тестових даних або пріоритезації тестових кейсів.
- Шаблон Декоратор (Decorator) - для динамічного додавання нової функціональності до існуючих об'єктів, наприклад, додавання нових полів або валідацій до об'єктів тестових кейсів або дефектів.

Підсумовуючи огляд підходів та технологій для створення інформаційної системи управління процесами тестування, можна зробити висновок, що вибір конкретного підходу, архітектури та технологій залежить від специфіки проєкту, вимог замовника, наявних ресурсів та кваліфікації команди розробки. При цьому важливо враховувати такі фактори, як масштабованість, гнучкість, безпека та продуктивність системи, а також можливість її інтеграції з іншими системами та інструментами в процесі розробки та експлуатації програмного забезпечення.

В контексті розробки інформаційної системи управління процесами тестування особливу увагу слід приділити вибору відповідних інструментів автоматизації тестування та безперервної інтеграції, які дозволяють значно підвищити ефективність та якість процесів тестування та виявлення дефектів. Також важливо забезпечити можливість інтеграції системи з існуючими системами управління вимогами, відслідковування дефектів та управління проєктами, щоб забезпечити цілісність та прозорість процесу розробки програмного забезпечення.

Окремої уваги заслуговує також вибір відповідних методів та підходів до організації процесу тестування в рамках обраної методології розробки (наприклад, Agile або DevOps). Зокрема, важливо забезпечити регулярне та автоматизоване тестування на всіх рівнях (модульне, інтеграційне, системне, приймальне) та етапах розробки (розробка, збірка, розгортання), а також налагодити ефективний процес управління та трекінгу дефектів з використанням відповідних інструментів та метрик якості.

Продовжуючи огляд підходів та технологій для створення інформаційної системи управління процесами тестування, варто також звернути увагу на деякі більш специфічні аспекти, які можуть вплинути на вибір конкретних рішень та інструментів.

Одним з таких аспектів є необхідність забезпечення безпеки та конфіденційності даних в процесі тестування. Особливо це стосується випадків, коли система управління процесами тестування інтегрується з іншими системами, які містять чутливі або конфіденційні дані (наприклад, системи управління персональними даними користувачів або фінансовими транзакціями). В таких випадках необхідно забезпечити відповідність системи вимогам стандартів безпеки (наприклад, ISO 27001, GDPR, PCI DSS тощо), а також реалізувати відповідні механізми захисту даних (наприклад, шифрування, аутентифікацію, авторизацію, аудит тощо).

Іншим важливим аспектом є необхідність забезпечення зручності використання та адаптивності системи до потреб різних груп користувачів (наприклад, тестувальників, розробників, менеджерів проєктів тощо). Це може вимагати реалізації різних інтерфейсів та функціональних можливостей для кожної групи користувачів, а також забезпечення можливості налаштування та персоналізації системи під потреби конкретних користувачів або команд.

Ще одним аспектом, який може вплинути на вибір технологій та інструментів для створення інформаційної системи управління процесами тестування, є необхідність інтеграції з іншими системами та інструментами в процесі розробки та експлуатації програмного забезпечення. Зокрема, система може потребувати інтеграції з:

- Системами управління вимогами (наприклад, Jira, Confluence, IBM Rational DOORS тощо) - для забезпечення зв'язку між вимогами та тестовими кейсами, а також для відслідковування покриття вимог тестами.
- Системами відслідковування дефектів (наприклад, Bugzilla, Redmine, Mantis тощо) - для забезпечення зв'язку між дефектами та тестовими кейсами, а також для відслідковування статусу та історії дефектів.
- Системами управління проєктами (наприклад, Microsoft Project, Primavera, Trello тощо) - для планування та відслідковування прогресу тестування в контексті загального плану проєкту.

- Системами безперервної інтеграції та доставки (наприклад, Jenkins, GitLab CI/CD, TeamCity тощо) - для автоматизації процесів збірки, тестування та розгортання програмного забезпечення.
- Системами моніторингу якості коду (наприклад, SonarQube, Coverity, Checkmarx тощо) - для автоматизованого аналізу якості коду та виявлення потенційних дефектів на ранніх етапах розробки.

Для вибору конкретних технологій та інструментів інтеграції необхідно враховувати такі фактори, як сумісність з існуючими системами, наявність відповідних API та протоколів обміну даними, а також вартість та складність реалізації інтеграції.

Нарешті, важливим аспектом вибору технологій та інструментів для створення інформаційної системи управління процесами тестування є також врахування перспектив розвитку та масштабування системи в майбутньому. Зокрема, при виборі технологій та архітектури системи необхідно враховувати такі фактори, як:

- Можливість горизонтального та вертикального масштабування системи для забезпечення її продуктивності та доступності при збільшенні кількості користувачів та обсягів даних.
- Можливість розширення функціональності системи за рахунок додавання нових модулів, плагінів або інтеграцій з іншими системами.
- Можливість міграції системи на інші платформи або технології в майбутньому, в тому числі з використанням хмарних технологій та сервісів.
- Наявність відповідної документації, навчальних матеріалів та спільноти розробників для обраних технологій та інструментів, що спростить підтримку та розвиток системи в майбутньому.

Для кращого розуміння переваг та недоліків різних технологій та інструментів для створення інформаційної системи управління процесами тестування, розглянемо порівняльну таблицю деяких з них:

Таблиця 1.2 - Порівняння технологій та інструментів для створення інформаційної системи управління процесами тестування

Технологія/ Інструмент	**Переваги**	**Недоліки**
Java	– Кросплатформеність – Широкий вибір фреймворків та бібліотек – Висока продуктивність та масштабованість[32] – Наявність великої спільноти розробників	– Відносно висока складність та вартість розробки – Необхідність встановлення та налаштування додаткового ПЗ (JDK, IDE тощо)
Python	– Простота та швидкість розробки – Широкий вибір бібліотек та фреймворків для різних задач – Підтримка різних парадигм програмування (ООП, функціональне) – Зручність для прототипування	– Відносно нижча продуктивність у порівнянні з компільованими мовами – Можливі проблеми з безпекою та якістю коду при неправильному використанні – Складність підтримки та рефакторингу великих проєктів
JavaScript/ TypeScript	– Швидкість та зручність розробки – Можливість розробки як серверної, так і клієнтської частини – Наявність великої кількості фреймворків та бібліотек – Підтримка асинхронного програмування	– Відносно нижча продуктивність та безпека у порівнянні з іншими мовами – Можливі проблеми з сумісністю та стабільністю при використанні різних версій та бібліотек

Продовження таблиці 1.2

Selenium	– Підтримка різних мов програмування та платформ – Можливість тестування на реальних браузерах та пристроях – Наявність зручного API та інструментів для запису та відтворення тестів	– Відносно висока складність налаштування та підтримки тестів – Залежність від версій та налаштувань браузерів – Можливі проблеми зі стабільністю та надійністю тестів
JMeter	– Можливість створення різних типів навантажувальних тестів – Підтримка різних протоколів (HTTP, JDBC, SOAP тощо) – Наявність зручного GUI та можливостей розширення функціональності	– Відносно висока складність налаштування та розширення функціональності – Обмежені можливості для інтеграції з іншими інструментами та системами – Потребує встановлення та налаштування додаткового ПЗ
Jenkins	– Підтримка різних мов та технологій розробки – Можливість автоматизації всього циклу розробки (CI/CD) – Наявність великої кількості плагінів та інтеграцій – Безкоштовність та відкритість	– Відносно висока складність налаштування та адміністрування – Потреба в додаткових ресурсах для розгортання та масштабування – Можливі проблеми з продуктивністю та безпекою при неправильному налаштуванні

Звичайно, наведена вище таблиця не є вичерпною і містить лише деякі з можливих технологій та інструментів. Кінцевий вибір конкретних рішень буде

залежати від специфічних вимог та обмежень кожного окремого проєкту, а також від наявних ресурсів та компетенцій команди розробки.

Підсумовуючи, можна зазначити, що створення ефективної інформаційної системи управління процесами тестування програмного забезпечення вимагає ретельного вибору відповідних підходів, технологій та інструментів, які б враховували як загальні принципи та кращі практики розробки програмного забезпечення, так і специфічні вимоги та обмеження предметної області.

При цьому важливо забезпечити не лише функціональність та ефективність системи, але й її безпечність, зручність використання, адаптивність та можливість подальшого розвитку та масштабування. Для цього необхідно враховувати такі фактори, як сумісність з існуючими системами та інструментами, можливість інтеграції з іншими системами, відповідність стандартам та регулятивним вимогам, а також перспективи розвитку обраних технологій та інструментів.

В наступних розділах роботи ми більш детально розглянемо конкретні проєктні рішення та підходи до розробки інформаційної системи управління процесами тестування, які базуються на розглянутих вище підходах та технологіях, а також врахують специфічні вимоги та обмеження предметної області.

РОЗДІЛ 2. ХАРАКТЕРИСТИКА СИСТЕМИ ТА МЕТОДІВ УПРАВЛІННЯ ПРОЦЕСАМИ ТЕСТУВАННЯ ТА ВИЯВЛЕННЯ ДЕФЕКТІВ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Структура і характеристика інформаційної системи у предметній області

Інформаційна система управління процесами тестування та виявлення дефектів програмного забезпечення є складною та багатокомпонентною системою, яка повинна забезпечувати ефективне виконання цілого ряду функцій та задач, пов'язаних з плануванням, організацією, проведенням та контролем якості процесів тестування на різних етапах життєвого циклу розробки програмного забезпечення.

Структура такої інформаційної системи повинна враховувати специфіку предметної області, а також забезпечувати можливість інтеграції з іншими системами та інструментами, які використовуються в процесі розробки та супроводу програмного забезпечення, такими як системи управління вимогами, системи управління проєктами, системи відслідковування дефектів, системи контролю версій тощо.

На найвищому рівні структуру інформаційної системи управління процесами тестування можна представити у вигляді кількох основних підсистем або модулів, кожен з яких відповідає за певний аспект функціональності системи.

Одним з ключових модулів такої системи є модуль управління тестовими активами, який відповідає за створення, зберігання, управління та використання різноманітних тестових артефактів, таких як тестові плани, тестові сценарії, тестові кейси, тестові дані, тестові скрипти тощо. Цей модуль повинен забезпечувати можливість створення та редагування тестових активів з використанням зручних та інтуїтивно зрозумілих інтерфейсів, а також їх зберігання в централізованому репозиторії з можливістю версіонування, пошуку та фільтрації за різними критеріями.

Крім того, модуль управління тестовими активами повинен підтримувати можливість імпорту та експорту тестових активів з/в різні формати та системи, а

також їх інтеграцію з іншими інструментами та системами, які використовуються в процесі тестування, такими як системи автоматизації тестування, системи управління тестовими даними тощо.

Іншим важливим модулем інформаційної системи управління процесами тестування є модуль планування та управління тестуванням, який відповідає за створення та управління тестовими планами, а також за організацію та контроль виконання тестових активностей на різних рівнях та етапах тестування.

Цей модуль повинен забезпечувати можливість створення та редагування тестових планів з використанням різних методологій та підходів до планування тестування, таких як ризик-орієнтоване тестування, тестування на основі вимог, тестування на основі моделей тощо. При цьому тестові плани повинні враховувати специфіку конкретного проєкту або продукту, а також вимоги та обмеження, які висуваються до процесу тестування з боку замовника, регуляторних органів або інших зацікавлених сторін.

Модуль планування та управління тестуванням повинен також забезпечувати можливість відслідковування та контролю виконання тестових активностей на різних рівнях та етапах тестування, включаючи модульне тестування, інтеграційне тестування, системне тестування, приймальне тестування тощо. При цьому система повинна надавати різноманітні інструменти та засоби для моніторингу та контролю процесу тестування, такі як дашборди, звіти, метрики якості тощо.

Ще одним ключовим модулем інформаційної системи управління процесами тестування є модуль управління дефектами, який відповідає за збір, класифікацію, аналіз та відслідковування дефектів, які виявляються в процесі тестування програмного забезпечення.

Цей модуль повинен забезпечувати можливість реєстрації дефектів з використанням стандартизованих шаблонів та класифікаторів, а також їх автоматичної маршрутизації та призначення відповідальним особам або командам згідно з встановленими правилами та процедурами. При цьому система повинна підтримувати можливість колаборативної роботи над дефектами з використанням різноманітних інструментів комунікації та координації, таких як коментарі, нотифікації, чати тощо.

Модуль управління дефектами повинен також забезпечувати можливість відслідковування життєвого циклу дефектів від моменту їх реєстрації до моменту закриття, включаючи такі стадії, як верифікація, аналіз кореневих причин, виправлення, ретестування тощо. При цьому система повинна надавати різноманітні інструменти та засоби для аналізу та візуалізації даних про дефекти, такі як діаграми Парето, тренди, розподіли за різними критеріями тощо.

Крім того, модуль управління дефектами повинен підтримувати можливість інтеграції з іншими системами та інструментами, які використовуються в процесі розробки та супроводу програмного забезпечення, такими як системи відслідковування помилок, системи управління вихідним кодом, системи безперервної інтеграції тощо. Це дозволяє забезпечити безшовний та ефективний процес виявлення, реєстрації та виправлення дефектів на всіх етапах життєвого циклу розробки програмного забезпечення.

Ще одним важливим модулем інформаційної системи управління процесами тестування є модуль автоматизації тестування, який відповідає за розробку, виконання та управління автоматизованими тестами з використанням різноманітних інструментів та фреймворків автоматизації.

Цей модуль повинен забезпечувати можливість створення та редагування автоматизованих тестових скриптів з використанням різних мов програмування та інструментів автоматизації, таких як Selenium, Appium, REST Assured тощо. При цьому система повинна підтримувати можливість інтеграції з іншими інструментами та системами, які використовуються в процесі розробки та тестування програмного забезпечення, такими як системи управління тестовими даними, системи безперервної інтеграції, системи віртуалізації тощо.

Модуль автоматизації тестування повинен також забезпечувати можливість планування та виконання автоматизованих тестів з використанням різних стратегій та підходів, таких як регресійне тестування, димове тестування, тестування продуктивності тощо. При цьому система повинна надавати різноманітні інструменти та засоби для управління та моніторингу виконання автоматизованих тестів, такі як тестові звіти, метрики якості, нотифікації про результати тестів тощо.

Нарешті, важливим модулем інформаційної системи управління процесами тестування є модуль аналітики та звітності, який відповідає за збір, обробку та візуалізацію різноманітних даних та метрик, пов'язаних з процесом тестування та якістю програмного забезпечення.

Цей модуль повинен забезпечувати можливість збору та консолідації даних з різних джерел та систем, які використовуються в процесі тестування, таких як системи управління тестовими активами, системи управління дефектами, системи автоматизації тестування тощо. При цьому система повинна підтримувати можливість обробки та трансформації даних з використанням різних алгоритмів та методів, таких як очищення даних, нормалізація, агрегація тощо.

Модуль аналітики та звітності повинен також забезпечувати можливість створення та візуалізації різноманітних звітів та дашбордів, які дозволяють відслідковувати та аналізувати ключові показники та метрики якості процесу тестування та програмного забезпечення, такі як покриття вимог тестами, щільність дефектів, ефективність виявлення дефектів, швидкість виправлення дефектів тощо. При цьому система повинна надавати гнучкі та зручні інструменти для налаштування та персоналізації звітів та дашбордів відповідно до потреб та вимог різних зацікавлених сторін, таких як менеджери проєктів, тестувальники, розробники, замовники тощо.

Окрім розглянутих вище основних модулів, інформаційна система управління процесами тестування повинна також включати ряд допоміжних модулів та компонентів, які забезпечують її ефективне функціонування та взаємодію з іншими системами та інструментами. До таких компонентів можна віднести:

- Модуль управління користувачами та правами доступу, який відповідає за реєстрацію, автентифікацію та авторизацію користувачів системи, а також за розподіл прав та ролей відповідно до встановлених політик та процедур.
- Модуль управління конфігурацією та налаштуваннями системи, який відповідає за зберігання та управління різноманітними параметрами та налаштуваннями системи, такими як шаблони документів, класифікатори дефектів, правила маршрутизації тощо.

- Модуль інтеграції та взаємодії з іншими системами, який відповідає за забезпечення обміну даними та синхронізації з іншими системами та інструментами, які використовуються в процесі розробки та супроводу програмного забезпечення, такими як системи управління вимогами, системи управління проєктами, системи відслідковування дефектів, системи контролю версій тощо.
- Модуль безпеки та аудиту, який відповідає за забезпечення конфіденційності, цілісності та доступності даних в системі, а також за реєстрацію та відслідковування дій користувачів та подій в системі з метою забезпечення відповідності регуляторним та нормативним вимогам.

Важливо зазначити, що наведена вище структура та характеристика інформаційної системи управління процесами тестування є узагальненою та може варіюватися в залежності від специфіки конкретного проєкту або організації, а також від обраних методологій, підходів та інструментів тестування.

Зокрема, в залежності від масштабу та складності проєкту, а також від рівня зрілості процесів тестування в організації, окремі модулі та компоненти системи можуть бути більш або менш розвиненими та інтегрованими між собою. Так, наприклад, в невеликих проєктах з обмеженими ресурсами та низьким рівнем автоматизації тестування, модуль автоматизації тестування може бути відсутнім або мати обмежену функціональність, в той час як в великих та складних проєктах з високим рівнем автоматизації цей модуль може бути одним з ключових та найбільш розвинених компонентів системи.

Крім того, на структуру та характеристику інформаційної системи управління процесами тестування можуть впливати також такі фактори, як:

- Тип та архітектура програмного забезпечення, що тестується (веб-додатки, мобільні додатки, десктопні додатки, вбудовані системи тощо).
- Методологія розробки програмного забезпечення, що використовується в проєкті (Waterfall, Agile, DevOps тощо).
- Стандарти та регуляторні вимоги, які висуваються до процесу тестування та якості програмного забезпечення (ISO, CMMI, FDA, GDPR тощо).

- Інструменти та технології, які використовуються в процесі тестування (мови програмування, фреймворки автоматизації, системи управління тестовими даними тощо).
- Організаційна структура та культура компанії, а також рівень кваліфікації та досвіду персоналу, залученого до процесу тестування.

Таким чином, проєктування та розробка інформаційної системи управління процесами тестування вимагає ретельного аналізу та врахування всіх цих факторів та особливостей, а також вибору відповідних підходів, методологій та інструментів, які б забезпечували максимальну ефективність та результативність процесу тестування в конкретних умовах та обставинах.

При цьому важливо забезпечити, щоб розроблена інформаційна система не лише відповідала поточним потребам та вимогам проєкту або організації, але й була здатна адаптуватися та розвиватися в майбутньому, враховуючи постійні зміни та виклики, які виникають в процесі розробки та супроводу програмного забезпечення.

Для цього необхідно закласти в архітектуру та дизайн системи принципи модульності, розширюваності та гнучкості, які дозволять додавати нові функціональні можливості та інтегрувати нові інструменти та технології без значних змін та модифікацій існуючих компонентів та підсистем.

Інформаційна система управління процесами тестування та виявлення дефектів програмного забезпечення є складною та багатокомпонентною системою, яка повинна забезпечувати ефективне виконання цілого ряду функцій та задач, пов'язаних з плануванням, організацією, проведенням та контролем якості процесів тестування на різних етапах життєвого циклу розробки програмного забезпечення.

Структура такої інформаційної системи повинна враховувати специфіку предметної області, а також забезпечувати можливість інтеграції з іншими системами та інструментами, які використовуються в процесі розробки та супроводу програмного забезпечення, такими як системи управління вимогами, системи управління проєктами, системи відслідковування дефектів, системи контролю версій тощо.

На найвищому рівні структуру інформаційної системи управління процесами тестування можна представити у вигляді кількох основних підсистем або модулів, кожен з яких відповідає за певний аспект функціональності системи. Основні модулі системи та їх призначення наведені в таблиці 2.1.

Таблиця 2.1 - Основні модулі інформаційної системи управління процесами тестування

Модуль	**Призначення**
Управління тестовими активами	Створення, зберігання, управління та використання тестових артефактів (тестові плани, сценарії, кейси, дані, скрипти)
Планування та управління тестуванням	Створення та управління тестовими планами, організація та контроль виконання тестових активностей
Управління дефектами	Збір, класифікація, аналіз та відслідковування дефектів, виявлених в процесі тестування
Автоматизація тестування	Розробка, виконання та управління автоматизованими тестами з використанням різних інструментів та фреймворків
Аналітика та звітність	Збір, обробка та візуалізація даних та метрик, пов'язаних з процесом тестування та якістю ПЗ

Окрім основних модулів, система також повинна включати ряд допоміжних компонентів, таких як:

- Модуль управління користувачами та правами доступу
- Модуль управління конфігурацією та налаштуваннями системи
- Модуль інтеграції та взаємодії з іншими системами
- Модуль безпеки та аудиту

Важливо відзначити, що наведена структура є узагальненою і може варіюватися залежно від специфіки конкретного проєкту, обраних методологій та інструментів тестування. На неї впливають такі фактори як тип ПЗ, що тестується,

методологія розробки, стандарти та регуляторні вимоги, організаційна структура компанії тощо.

Для забезпечення ефективності та адаптивності системи в майбутньому, в її архітектуру мають бути закладені принципи модульності, розширюваності та гнучкості. Це дозволить додавати нові можливості та інтегрувати нові інструменти без суттєвих змін існуючих компонентів.

Таким чином, раціонально спроєктована інформаційна система управління процесами тестування, яка враховує всі ключові аспекти та особливості предметної області, здатна суттєво підвищити ефективність та результативність процесу тестування програмного забезпечення.

2.2 Огляд методів та моделей в інформаційних системах та технологіях, пов’язаних з управлінням процесами тестування та виявлення дефектів ПЗ

Управління процесами тестування та виявлення дефектів програмного забезпечення є складним та багатогранним завданням, яке вимагає використання різноманітних методів, моделей та технологій в рамках відповідних інформаційних систем. Розглянемо деякі з найбільш поширених та ефективних підходів у цій сфері.

Одним з ключових методів в управлінні процесами тестування є ризик-орієнтоване тестування (Risk-Based Testing, RBT). Цей підхід базується на ідентифікації, аналізі та пріоритезації ризиків, пов'язаних з якістю програмного забезпечення, та плануванні тестових активностей таким чином, щоб забезпечити максимальне покриття найбільш критичних ризиків.

В рамках RBT використовуються різні техніки оцінки ризиків, такі як FMEA (Failure Mode and Effects Analysis), FTA (Fault Tree Analysis), ETA (Event Tree Analysis) тощо. Ці техніки дозволяють ідентифікувати потенційні відмови та дефекти програмного забезпечення, оцінити їх вплив та ймовірність виникнення, а також визначити необхідні тестові заходи для їх попередження або виявлення.

Інформаційні системи, які підтримують ризик-орієнтоване тестування, повинні забезпечувати можливості для збору, зберігання та аналізу даних про ризики, а також для планування та відслідковування тестових активностей на

основі цих даних. Такі системи можуть включати в себе модулі для управління вимогами, управління тестовими активами, управління дефектами, а також аналітичні інструменти для оцінки та візуалізації ризиків.

Іншим важливим методом в управлінні процесами тестування є тестування на основі моделей (Model-Based Testing, MBT). Цей підхід передбачає створення формальних моделей системи, які описують її поведінку, функціональність та властивості, та використання цих моделей для автоматичної генерації тестових випадків та сценаріїв.

Моделі можуть бути представлені у різних формах, таких як діаграми станів, діаграми активності, діаграми послідовності, таблиці прийняття рішень тощо. Вони можуть бути створені з використанням спеціалізованих мов моделювання, таких як UML (Unified Modeling Language), SysML (Systems Modeling Language), AADL (Architecture Analysis and Design Language) та інших.

Інформаційні системи для підтримки MBT повинні забезпечувати можливості для створення, редагування та зберігання моделей, а також для автоматичної генерації тестових артефактів (тестових випадків, сценаріїв, скриптів) на основі цих моделей. Такі системи можуть включати в себе графічні редактори моделей, генератори тестів, компонентами для виконання та аналізу результатів тестів.

Ще одним підходом, який активно використовується в управлінні процесами тестування, є тестування на основі досвіду (Experience-Based Testing, EBT). Цей підхід базується на використанні знань та досвіду тестувальників для планування та проведення тестових активностей.

В рамках EBT застосовуються різні техніки, такі як інтуїтивне тестування, дослідницьке тестування, тестування за аналогією, чек-лист тестування тощо. Ці техніки дозволяють виявляти дефекти та проблеми, які можуть бути пропущені при формальному тестуванні, а також забезпечувати більш ефективне використання обмежених ресурсів тестування.

Інформаційні системи для підтримки EBT повинні надавати тестувальникам зручні інструменти для документування та обміну знаннями та досвідом, такі як бази знань, вікі-системи, форуми, чати тощо. Вони також повинні забезпечувати

можливості для збору та аналізу даних про результати тестування, виявлені дефекти, ефективність різних тестових технік та підходів.

В управлінні процесами виявлення та усунення дефектів програмного забезпечення важливу роль відіграють системи відслідковування дефектів (Bug Tracking Systems, BTS). Ці системи призначені для реєстрації, класифікації, призначення, відслідковування та управління дефектами та проблемами, виявленими на різних етапах життєвого циклу програмного забезпечення.

Сучасні BTS надають різноманітні функції та можливості, такі як:

- Реєстрація дефектів з детальним описом, кроками відтворення, очікуваними та фактичними результатами, скріншотами та іншими артефактами.
- Класифікація дефектів за різними критеріями (тип, серйозність, пріоритет, компонент, версія тощо).
- Призначення дефектів відповідальним особам або командам для аналізу та виправлення.
- Відслідковування життєвого циклу дефектів (статуси, коментарі, зміни, зв'язки з іншими артефактами).
- Генерація звітів та метрик щодо кількості, динаміки та розподілу дефектів за різними параметрами.
- Інтеграція з іншими системами (системи управління версіями, системи безперервної інтеграції, системи управління проектами тощо).

Прикладами популярних BTS є Jira, Bugzilla, Redmine, Mantis, Google Issue Tracker та інші. Вибір конкретної BTS залежить від масштабу та специфіки проекту, а також від наявних ресурсів та інфраструктури.

Окрім розглянутих вище методів та систем, в управлінні процесами тестування та виявлення дефектів програмного забезпечення використовуються також різноманітні методології та фреймворки, такі як:

- TMMi (Test Maturity Model Integration) - модель зрілості процесів тестування, яка дозволяє оцінювати та покращувати якість цих процесів.
- ISTQB (International Software Testing Qualifications Board) - міжнародна некомерційна організація, яка розробляє та підтримує стандарти та сертифікації в області тестування програмного забезпечення.

- TMAP (Test Management Approach) - структурований підхід до управління тестуванням, який охоплює всі аспекти процесу тестування та забезпечує його відповідність бізнес-цілям.
- Agile Testing - підхід до тестування в рамках гнучких методологій розробки, який передбачає активну співпрацю тестувальників з іншими членами команди та ітеративне тестування протягом всього життєвого циклу розробки.
- Exploratory Testing - підхід до тестування, який базується на одночасному вивченні, проектуванні та виконанні тестів, а також на креативності та досвіді тестувальника[19].

Окремо слід відзначити також інструменти автоматизації тестування, які відіграють все більшу роль в сучасних процесах розробки програмного забезпечення. Ці інструменти дозволяють значно підвищити ефективність та швидкість тестування, особливо для великих та комплексних систем.

Найбільш поширеними інструментами автоматизації тестування є:

- Selenium - фреймворк з відкритим кодом для автоматизації тестування веб-додатків.
- Appium - інструмент для автоматизації тестування нативних, гібридних та веб-додатків на мобільних платформах (iOS, Android).
- JMeter - інструмент для проведення навантажувального та стрес-тестування веб-додатків та сервісів.
- TestComplete - комерційна платформа для автоматизації тестування настільних, веб та мобільних додатків.
- Robot Framework - фреймворк з відкритим кодом для автоматизації тестування, який підтримує різні типи додатків та технології.

Впровадження та ефективне використання цих інструментів в рамках інформаційних систем управління процесами тестування вимагає відповідної інфраструктури, кваліфікації персоналу та налагодженої взаємодії між різними командами та фахівцями (тестувальниками, розробниками, DevOps-інженерами тощо).

Підсумовуючи, можна зазначити, що сучасні інформаційні системи та технології надають широкий спектр методів, моделей та інструментів для ефективного управління процесами тестування та виявлення дефектів програмного забезпечення. Вибір та застосування конкретних підходів та засобів залежить від специфіки проекту, доступних ресурсів та рівня зрілості процесів в організації.

При цьому ключовими факторами успіху є комплексність та інтегрованість підходу, який охоплює всі аспекти процесу тестування (планування, проектування, виконання, звітність, управління дефектами тощо), а також постійне вдосконалення та адаптація процесів відповідно до зміни вимог, технологій та умов функціонування програмного забезпечення.

Лише впровадження окремих методів чи інструментів не може гарантувати успіх в управлінні процесами тестування та виявлення дефектів. Важливо забезпечити системний та комплексний підхід, який враховує всі аспекти та рівні процесу тестування, а також забезпечує ефективну взаємодію та комунікацію між усіма зацікавленими сторонами.

Одним з ключових елементів такого підходу є формування та розвиток культури якості в організації. Це передбачає не лише впровадження формальних процесів та практик тестування, але й зміну мислення та ставлення до якості на всіх рівнях - від вищого керівництва до окремих розробників та тестувальників.

В рамках культури якості тестування розглядається не як окрема фаза чи активність, а як невід'ємна частина всього процесу розробки, яка інтегрована з іншими активностями (аналіз вимог, проектування, кодування, розгортання тощо) та виконується на постійній основі. При цьому відповідальність за якість розподіляється між усіма членами команди, а не покладається лише на тестувальників.

Для підтримки та розвитку культури якості необхідно забезпечити відповідну організаційну структуру, процеси та практики, які стимулюють співпрацю, обмін знаннями та постійне вдосконалення. Це може включати в себе такі елементи як:

- Крос-функціональні команди, які об'єднують фахівців з різних областей (розробка, тестування, аналіз, DevOps тощо) та працюють разом над спільними цілями.
- Регулярні зустрічі та огляди коду, на яких обговорюються питання якості, виявляються та усуваються потенційні проблеми та дефекти.
- Ретроспективи та постмортеми, на яких аналізуються результати та досвід попередніх ітерацій або релізів, та визначаються напрямки для покращення процесів та практик.
- Програми навчання та сертифікації, які дозволяють розвивати та підтверджувати компетенції фахівців в області тестування та якості.
- Системи мотивації та заохочення, які стимулюють та винагороджують внесок у забезпечення якості на всіх рівнях організації.

Інформаційні системи управління процесами тестування повинні підтримувати та доповнювати ці організаційні та культурні аспекти, надаючи необхідні інструменти, дані та аналітику для прийняття обґрунтованих рішень та постійного вдосконалення процесів.

Наприклад, такі системи можуть включати в себе модулі для збору та аналізу метрик якості, таких як щільність дефектів, відсоток автоматизації тестування, час виявлення та усунення дефектів тощо. Ці метрики можуть бути візуалізовані у вигляді інформаційних панелей (dashboards) та звітів, доступних для різних зацікавлених сторін (менеджерів, тестувальників, розробників тощо).

В таблиці 2.2 наведено приклади деяких ключових метрик якості та їх опис:

Метрика	**Опис**
Щільність дефектів	Кількість дефектів на одиницю розміру коду (наприклад, на 1000 рядків коду). Показує загальну якість коду та ефективність процесу тестування.
Відсоток автоматизації тестування	Відношення автоматизованих тестів до загальної кількості тестів. Показує рівень автоматизації та ефективність процесу тестування.

Продовження таблиці 2.2

Час виявлення дефектів	Середній час між появою дефекту та його виявленням. Показує ефективність процесу тестування та якість тестового покриття.
Час усунення дефектів	Середній час між виявленням дефекту та його усуненням. Показує ефективність процесу усунення дефектів та співпраці між тестувальниками та розробниками.
Відсоток прийнятих користувачами дефектів	Відношення дефектів, виявлених користувачами після релізу, до загальної кількості дефектів. Показує ефективність процесу тестування та відповідність продукту очікуванням користувачів.

Інформаційні системи управління процесами тестування можуть також включати в себе інструменти для аналізу та оптимізації процесів на основі зібраних даних та метрик. Наприклад, за допомогою методів process mining можна виявляти вузькі місця, затримки та відхилення в процесах тестування, а також моделювати та порівнювати різні сценарії оптимізації.

Отже, ефективне управління процесами тестування та виявлення дефектів вимагає комплексного підходу, який поєднує в собі відповідні методи, моделі, інструменти та практики, а також враховує організаційні, культурні та людські аспекти забезпечення якості програмного забезпечення.

Інформаційні системи відіграють важливу роль в цьому підході, надаючи необхідну підтримку, автоматизацію, дані та аналітику для прийняття обґрунтованих рішень та постійного вдосконалення процесів. При цьому вибір та впровадження конкретних систем та технологій повинен здійснюватись з урахуванням специфіки організації, її цілей, ресурсів та рівня зрілості процесів.

РОЗДІЛ 3. РОЗРОБЛЕННЯ ПРОЄКТНИХ РІШЕНЬ ДЛЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ УПРАВЛІННЯ ТЕСТУВАННЯМ ТА ВИЯВЛЕННЯМ ДЕФЕКТІВ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Проєктування бази даних

Проєктування бази даних є одним з ключових етапів розробки інформаційної системи управління тестуванням та виявленням дефектів програмного забезпечення. База даних служить основою для зберігання, організації та управління всією інформацією, необхідною для ефективного функціонування системи.

Процес проєктування бази даних починається з аналізу вимог до системи та визначення сутностей, які будуть представлені в базі даних. У контексті системи управління тестуванням та виявленням дефектів, основними сутностями є проєкти, тестові сценарії, тестові плани, дефекти, користувачі та інші пов'язані з ними дані.

Для забезпечення цілісності та узгодженості даних, а також для уникнення надлишковості та аномалій, при проєктуванні бази даних використовується процес нормалізації. Нормалізація передбачає розбиття таблиць на менші, більш атомарні таблиці, які відповідають певним нормальним формам (1НФ, 2НФ, 3НФ тощо). Це дозволяє зменшити надлишковість даних та забезпечити їх цілісність.

У нашій інформаційній системі база даних спроєктована з використанням реляційної моделі. Реляційна модель базується на концепції таблиць (відношень), де кожна таблиця представляє певну сутність, а стовпці таблиці відповідають атрибутам цієї сутності[30]. Зв'язки між таблицями встановлюються за допомогою первинних та зовнішніх ключів, що забезпечує цілісність даних та дозволяє виконувати ефективні операції вибірки та маніпулювання даними.

Для проєктування бази даних було використано методологію ER (Entity-Relationship) моделювання[7]. ER-модель дозволяє візуально представити сутності, їх атрибути та зв'язки між ними[8]. Для створення ER-діаграми використано програмне забезпечення для моделювання баз даних, таке як MySQL Workbench або ERwin Data Modeler.

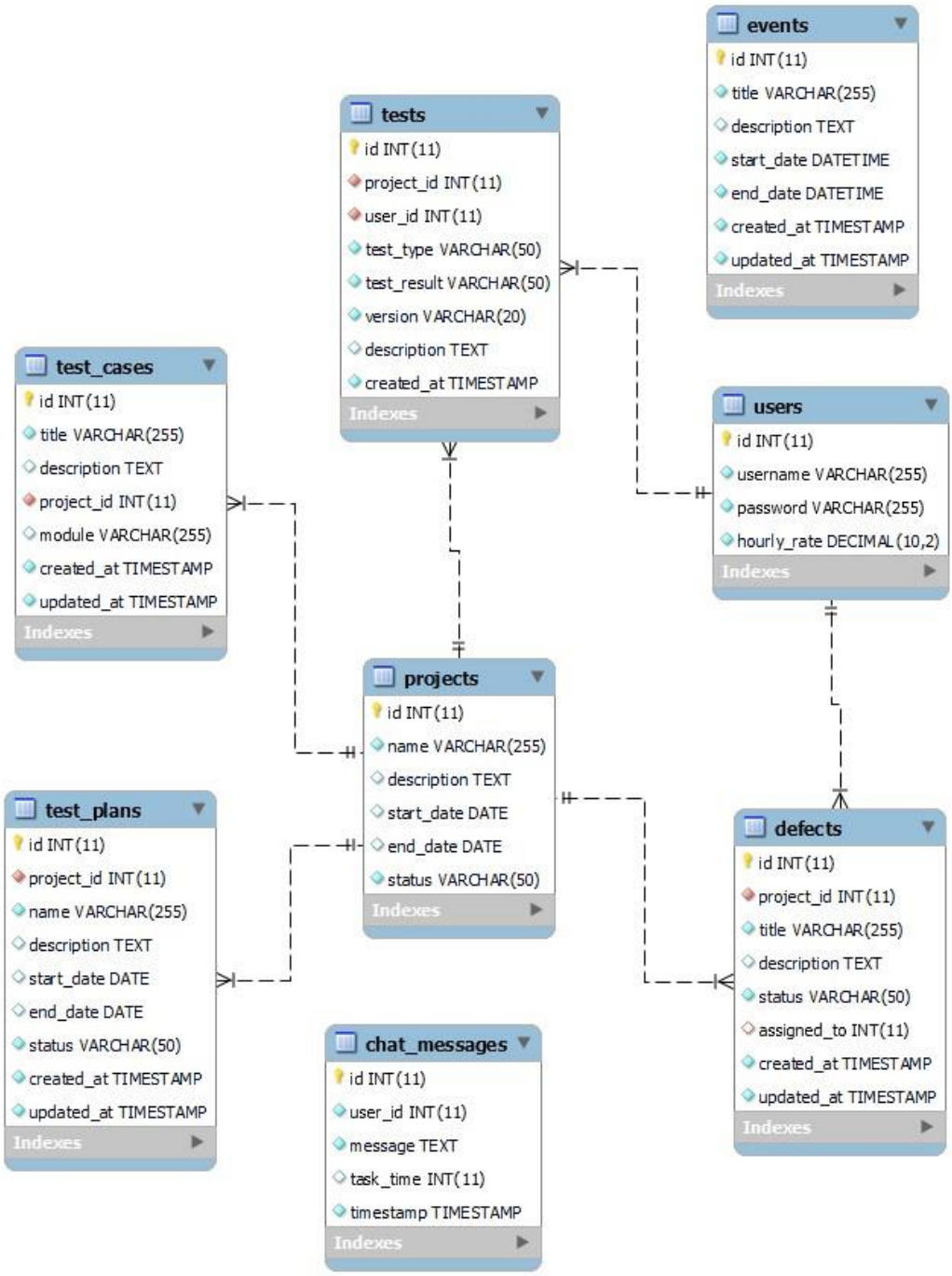

Рисунок 3.1 – діаграма бази даних

На ER-діаграмі представлені основні сутності системи, такі як проєкти (Projects), тестові сценарії (Test Cases), тестові плани (Test Plans), дефекти (Defects), користувачі (Users) та інші. Кожна сутність має свої атрибути, які визначають її властивості. Наприклад, сутність "Проєкти" може мати атрибути "Назва проєкту", "Опис проєкту", "Дата початку" та "Дата завершення".

Зв'язки між сутностями на ER-діаграмі представлені лініями, які з'єднують відповідні сутності. Наприклад, між сутностями "Проєкти" та "Тестові сценарії" може бути зв'язок "один-до-багатьох", що означає, що один проєкт може мати

багато тестових сценаріїв, але кожен тестовий сценарій належить лише одному проєкту.

Після створення ER-моделі, вона трансформується у фізичну модель бази даних, яка враховує особливості обраної системи управління базами даних (СУБД). У нашому випадку, для реалізації бази даних обрано СУБД MySQL. Фізична модель включає детальний опис таблиць, їх полів, типів даних, обмежень та зв'язків між таблицями.

При проєктуванні бази даних також враховуються вимоги до продуктивності та масштабованості системи. Для забезпечення оптимальної продуктивності, створюються необхідні індекси на полях, які часто використовуються для пошуку та фільтрації даних. Це дозволяє значно прискорити виконання запитів до бази даних.

Для забезпечення безпеки та конфіденційності даних, при проєктуванні бази даних враховуються механізми контролю доступу та авторизації користувачів. Кожному користувачу призначаються відповідні права доступу до таблиць та операцій, які вони можуть виконувати.

Крім того, при проєктуванні бази даних враховуються вимоги до резервного копіювання та відновлення даних. Розробляється стратегія резервного копіювання, яка включає регулярне створення повних та інкрементальних резервних копій бази даних. Це дозволяє забезпечити можливість відновлення даних у випадку збоїв або втрати інформації.

Після завершення проєктування бази даних, створюється набір SQL-скриптів для створення таблиць, встановлення зв'язків, додавання обмежень та індексів. Ці скрипти використовуються для ініціалізації бази даних при розгортанні інформаційної системи.

Таким чином, проєктування бази даних є важливим етапом розробки інформаційної системи управління тестуванням та виявленням дефектів програмного забезпечення. Правильно спроєктована база даних забезпечує ефективне зберігання, організацію та доступ до даних, що є критичним для успішного функціонування системи. ER-моделювання та нормалізація даних

дозволяють створити структуровану та цілісну базу даних, яка відповідає вимогам системи та забезпечує її надійність та продуктивність.

3.2 Проєктування засобів інтелектуального аналізу даних

Інтелектуальний аналіз даних (Data Mining) є важливою складовою інформаційної системи управління тестуванням та виявленням дефектів програмного забезпечення. Він дозволяє виявляти приховані закономірності, тенденції та взаємозв'язки в даних, що накопичуються в процесі тестування та розробки програмного забезпечення. Проєктування засобів інтелектуального аналізу даних має на меті створення ефективних інструментів та методів для отримання цінних знань та прийняття обґрунтованих рішень.

Процес проєктування засобів інтелектуального аналізу даних починається з визначення цілей та завдань аналізу. У контексті системи управління тестуванням та виявленням дефектів, основними цілями можуть бути:

1. Виявлення патернів та закономірностей у виникненні дефектів: Аналіз даних про дефекти може допомогти виявити найбільш поширені типи дефектів, їх розподіл за модулями або компонентами системи, а також зв'язки між різними факторами, такими як версія програмного забезпечення, платформа, конфігурація тощо.
2. Прогнозування ризиків та планування тестування: Інтелектуальний аналіз даних може використовуватися для прогнозування потенційних ризиків та проблемних областей в програмному забезпеченні на основі історичних даних про дефекти та результати попередніх тестувань. Це дозволяє оптимізувати планування тестування та розподіл ресурсів.
3. Оцінка ефективності тестування: Аналіз даних про результати тестування, покриття коду, час виконання тестів та інші показники може допомогти оцінити ефективність процесу тестування та виявити області для вдосконалення.
4. Виявлення аномалій та відхилень: Інтелектуальний аналіз даних може використовуватися для виявлення нетипових або підозрілих патернів у

даних, таких як раптові зміни в кількості дефектів, незвичайні комбінації факторів або відхилення від очікуваних трендів.

Для досягнення цих цілей, при проєктуванні засобів інтелектуального аналізу даних використовуються різноманітні методи та алгоритми. Серед них:

1. Класифікація: Методи класифікації дозволяють розподіляти дані на заздалегідь визначені категорії або класи. Наприклад, класифікація дефектів за типом (функціональні, продуктивності, безпеки тощо) або за критичністю (низька, середня, висока).
2. Кластеризація: Кластеризація дозволяє групувати дані на основі їх подібності або близькості за певними ознаками. Це може бути корисним для виявлення груп схожих дефектів або тестових випадків.
3. Асоціативні правила: Методи асоціативних правил дозволяють виявляти зв'язки та залежності між різними елементами даних. Наприклад, можна виявити, що певні комбінації факторів (версія ПЗ, операційна система, конфігурація) частіше призводять до виникнення певних типів дефектів.
4. Регресійний аналіз: Регресійний аналіз використовується для прогнозування значень однієї змінної на основі значень інших змінних. Наприклад, можна будувати моделі для прогнозування кількості дефектів або часу, необхідного для їх виправлення, на основі різних факторів.
5. Аналіз часових рядів: Методи аналізу часових рядів дозволяють виявляти закономірності та тренди в даних, що змінюються з часом. Це може бути корисним для відстеження динаміки виникнення дефектів або прогнозування майбутніх тенденцій.

При проєктуванні засобів інтелектуального аналізу даних, важливо враховувати якість та підготовку вхідних даних. Дані повинні бути попередньо оброблені, очищені від шуму та аномалій, а також приведені до єдиного формату. Також необхідно провести попередній аналіз даних, щоб зрозуміти їх структуру, розподіл та основні статистичні характеристики.

Для реалізації засобів інтелектуального аналізу даних використовуються різноманітні інструменти та бібліотеки[9]. Мови програмування, такі як Python або R, надають широкий спектр бібліотек для машинного навчання та аналізу даних,

наприклад, scikit-learn, TensorFlow, pandas, matplotlib тощо[10]. Також можуть використовуватися спеціалізовані інструменти для інтелектуального аналізу даних, такі як RapidMiner, KNIME, Weka або SAS Enterprise Miner[40].

Важливим аспектом проєктування засобів інтелектуального аналізу даних є візуалізація результатів. Візуалізація дозволяє представити отримані знання та закономірності у зрозумілій та наочній формі, що полегшує їх інтерпретацію та використання. Для візуалізації можуть використовуватися різноманітні графіки, діаграми, гістограми, теплові карти тощо.

Крім того, при проєктуванні засобів інтелектуального аналізу даних враховуються вимоги до продуктивності та масштабованості. Обробка великих обсягів даних може потребувати значних обчислювальних ресурсів та оптимізації алгоритмів. Тому, при необхідності, розглядаються можливості розподіленої обробки даних, використання технологій Big Data та хмарних обчислень.

Для забезпечення інтеграції засобів інтелектуального аналізу даних з іншими компонентами інформаційної системи, проєктуються відповідні інтерфейси та механізми взаємодії. Результати аналізу повинні бути доступними для інших модулів системи, таких як звітність, візуалізація даних або підтримка прийняття рішень.

Після проєктування засобів інтелектуального аналізу даних, проводиться їх тестування та валідація на реальних даних. Це дозволяє перевірити ефективність та точність розроблених моделей та алгоритмів, а також внести необхідні корективи та вдосконалення.

Таким чином, проєктування засобів інтелектуального аналізу даних є важливим етапом розробки інформаційної системи управління тестуванням та виявленням дефектів програмного забезпечення. Правильно спроєктовані засоби дозволяють отримувати цінні знання та закономірності з накопичених даних, що сприяє прийняттю обґрунтованих рішень, оптимізації процесу тестування та підвищенню якості програмного забезпечення. Використання сучасних методів та інструментів інтелектуального аналізу даних, а також врахування вимог до якості даних, продуктивності та масштабованості, забезпечують ефективність та надійність розроблених засобів[38].

3.3 Розробка програмного забезпечення

Розробка програмного забезпечення є ключовим етапом у створенні інформаційної системи управління тестуванням та виявленням дефектів програмного забезпечення. На цьому етапі відбувається безпосередня реалізація функціональних можливостей системи, її архітектури та інтерфейсів користувача.

Для розробки нашої інформаційної системи було обрано мову програмування PHP та фреймворк Laravel[27]. PHP є широко використовуваною мовою програмування для веб-розробки, яка надає потужні можливості для створення динамічних веб-додатків[14]. Laravel, в свою чергу, є популярним PHP-фреймворком, який забезпечує структурований та ефективний підхід до розробки, дотримуючись принципів MVC (Model-View-Controller) та надаючи багато вбудованих функцій та інструментів.

Розробка програмного забезпечення для нашої системи включала створення різних модулів та функцій, які забезпечують реалізацію основних функціональних вимог. Розглянемо детальніше розроблені модулі та функції:

1. Модуль управління проектами:
    - Функція створення нового проекту: дозволяє користувачам створювати нові проекти, вказуючи назву, опис, дати початку та завершення, а також інші необхідні деталі.
    - Функція редагування проекту: дозволяє користувачам редагувати інформацію про існуючі проекти, оновлюючи їх назву, опис, дати та інші параметри.
    - Функція видалення проекту: дозволяє користувачам видаляти проекти, які більше не потрібні.
    - Функція перегляду списку проектів: відображає список усіх доступних проектів з можливістю фільтрації та сортування.
2. Модуль управління тестовими сценаріями:
    - Функція створення нового тестового сценарію: дозволяє користувачам створювати нові тестові сценарії, вказуючи назву, опис, кроки виконання, очікувані результати та інші необхідні дані.

- Функція редагування тестового сценарію: дозволяє користувачам редагувати існуючі тестові сценарії, оновлюючи їх назву, опис, кроки та інші параметри.
- Функція видалення тестового сценарію: дозволяє користувачам видаляти тестові сценарії, які більше не потрібні.
- Функція перегляду списку тестових сценаріїв: відображає список усіх доступних тестових сценаріїв для вибраного проекту з можливістю фільтрації та сортування.

3. Модуль управління тестовими планами:
    - Функція створення нового тестового плану: дозволяє користувачам створювати нові тестові плани, вказуючи назву, опис, дати початку та завершення, а також вибираючи тестові сценарії, які будуть включені до плану.
    - Функція редагування тестового плану: дозволяє користувачам редагувати існуючі тестові плани, оновлюючи їх назву, опис, дати та список включених тестових сценаріїв.
    - Функція видалення тестового плану: дозволяє користувачам видаляти тестові плани, які більше не потрібні.
    - Функція перегляду списку тестових планів: відображає список усіх доступних тестових планів для вибраного проекту з можливістю фільтрації та сортування.
4. Модуль управління дефектами:
    - Функція створення нового дефекту: дозволяє користувачам створювати нові дефекти, вказуючи назву, опис, пріоритет, статус, кроки для відтворення та інші необхідні дані.
    - Функція редагування дефекту: дозволяє користувачам редагувати існуючі дефекти, оновлюючи їх назву, опис, пріоритет, статус та інші параметри.
    - Функція видалення дефекту: дозволяє користувачам видаляти дефекти, які більше не актуальні.

- Функція перегляду списку дефектів: відображає список усіх зареєстрованих дефектів для вибраного проекту з можливістю фільтрації та сортування.

5. Модуль генерації звітів:
    - Функція генерації звіту про виконання тестових планів: формує звіт, який відображає статистику виконання тестових планів, включаючи кількість успішних та неуспішних тестів, відсоток покриття вимог тощо.
    - Функція генерації звіту про дефекти: формує звіт, який відображає статистику зареєстрованих дефектів, їх розподіл за пріоритетами, статусами та іншими параметрами.
    - Функція генерації звіту про прогрес тестування: формує звіт, який відображає прогрес виконання тестових активностей у часі, включаючи кількість виконаних тестів, знайдених дефектів тощо.
6. Модуль аналітики та візуалізації даних:
    - Функція відображення загальної статистики: відображає загальну статистику по проектах, тестових сценаріях, дефектах та іншим ключовим показникам.
    - Функція відображення графіків та діаграм: будує різноманітні графіки та діаграми для візуалізації даних, таких як розподіл дефектів за пріоритетами, динаміка виявлення дефектів у часі тощо.
    - Функція інтерактивних інформаційних панелей: надає користувачам можливість створювати та налаштовувати інтерактивні інформаційні панелі з вибраними показниками та графіками.
7. Модуль управління користувачами та правами доступу:
    - Функція реєстрації нових користувачів: дозволяє адміністраторам створювати нові облікові записи користувачів з відповідними правами доступу.
    - Функція редагування профілю користувача: дозволяє користувачам редагувати свої особисті дані, такі як ім'я, електронна пошта тощо.

- Функція керування правами доступу: дозволяє адміністраторам призначати та змінювати права доступу користувачів до різних модулів та функцій системи.

Крім розробки основних модулів та функцій, було приділено увагу створенню зручного та інтуїтивно зрозумілого інтерфейсу користувача. Використано сучасні технології верстки, такі як HTML, CSS та JavaScript, для забезпечення адаптивності та крос-браузерної сумісності інтерфейсу.

Для забезпечення безпеки та захисту даних, реалізовано механізми автентифікації та авторизації користувачів. Використано методи шифрування паролів, перевірки прав доступу та захисту від поширених веб-вразливостей, таких як SQL-ін'єкції та міжсайтовий скриптинг (XSS).

Також, приділено увагу оптимізації продуктивності та масштабованості системи. Використано техніки кешування даних, оптимізації запитів до бази даних та асинхронної обробки завдань для забезпечення швидкого відгуку системи навіть при обробці великих обсягів даних.

Для забезпечення якості розробленого програмного забезпечення, проведено ретельне тестування на різних рівнях[21]. Реалізовано модульні тести для перевірки коректності роботи окремих функцій та компонентів, інтеграційні тести для перевірки взаємодії між модулями, а також системні тести для перевірки загальної функціональності та продуктивності системи[22].

Таким чином, розробка програмного забезпечення для інформаційної системи управління тестуванням та виявленням дефектів програмного забезпечення включала створення різноманітних модулів та функцій, які забезпечують реалізацію основних функціональних вимог. Використання сучасних технологій, дотримання принципів безпеки та оптимізації, а також проведення ретельного тестування, дозволило створити надійне та ефективне програмне рішення для підтримки процесів тестування та управління якістю програмного забезпечення.

3.4 Визначення технічного забезпечення

Визначення технічного забезпечення є важливим етапом у розробці інформаційної системи управління тестуванням та виявленням дефектів програмного забезпечення. На цьому етапі визначаються апаратні та програмні компоненти, необхідні для ефективного функціонування системи, а також вимоги до інфраструктури та мережевого оточення.

Для забезпечення надійної та продуктивної роботи нашої інформаційної системи, було визначено наступні вимоги до технічного забезпечення:

1. Серверне обладнання:
    - Сервер додатків: для розміщення та виконання серверної частини інформаційної системи необхідний потужний сервер з достатніми обчислювальними ресурсами. Рекомендовано використовувати сервер з процесором Intel Xeon або аналогічним, з частотою не менше 2,5 ГГц та кількістю ядер не менше 8.
    - Сервер бази даних: для зберігання та обробки даних системи необхідний окремий сервер бази даних. Рекомендовано використовувати сервер з процесором Intel Xeon або аналогічним, з частотою не менше 2,5 ГГц та кількістю ядер не менше 8, а також з достатнім обсягом оперативної пам'яті (не менше 32 ГБ) для забезпечення швидкої обробки запитів.
    - Дисковий простір: для зберігання даних, логів та резервних копій необхідний достатній обсяг дискового простору. Рекомендовано використовувати високопродуктивні твердотільні накопичувачі (SSD) з ємністю не менше 1 ТБ.
2. Мережеве обладнання:
    - Маршрутизатори та комутатори: для забезпечення надійного та швидкого мережевого з'єднання між серверами та клієнтськими робочими станціями необхідно використовувати високопродуктивні маршрутизатори та комутатори з підтримкою гігабітних швидкостей передачі даних[11, 12].

- Міжмережевий екран: для захисту мережі та серверів від неавторизованого доступу та потенційних загроз безпеки необхідно використовувати міжмережевий екран з можливістю налаштування правил фільтрації трафіку та виявлення вторгнень.

3. Клієнтські робочі станції:
    - Комп'ютери користувачів: для доступу до інформаційної системи користувачам необхідні робочі станції з достатніми обчислювальними ресурсами. Рекомендовано використовувати комп'ютери з процесором Intel Core i5 або аналогічним, з частотою не менше 2,5 ГГц та обсягом оперативної пам'яті не менше 8 ГБ.
    - Операційна система: на клієнтських робочих станціях рекомендовано використовувати сучасну операційну систему, таку як Windows 10 або macOS, з останніми оновленнями та патчами безпеки.
    - Веб-браузер: для доступу до веб-інтерфейсу інформаційної системи користувачам необхідний сучасний веб-браузер, такий як Google Chrome, Mozilla Firefox або Microsoft Edge, з підтримкою останніх веб-стандартів та технологій.
4. Програмне забезпечення:
    - Операційні системи серверів: на серверах рекомендовано використовувати стабільну та безпечну операційну систему, таку як Ubuntu Server або CentOS, з регулярними оновленнями та патчами безпеки.
    - Веб-сервер: для обробки HTTP-запитів та обслуговування веб-додатку необхідний веб-сервер, такий як Apache або Nginx, налаштований відповідно до вимог продуктивності та безпеки.
    - Сервер бази даних: для зберігання та управління даними системи необхідна система управління базами даних (СУБД), така як MySQL або PostgreSQL, з оптимальними налаштуваннями продуктивності та безпеки.
    - Мова програмування та фреймворки: для розробки серверної частини системи використовується мова програмування PHP та фреймворк

Laravel, які повинні бути встановлені та налаштовані на сервері додатків[13].

5. Резервне копіювання та відновлення:
    - Система резервного копіювання: для забезпечення збереження даних та можливості відновлення у випадку збоїв або втрати даних необхідно реалізувати систему регулярного резервного копіювання. Рекомендовано використовувати програмне забезпечення для резервного копіювання, таке як Veeam або Acronis, з можливістю автоматичного створення розкладу резервних копій.
    - Сховище резервних копій: резервні копії повинні зберігатися на окремому сховищі, бажано на іншій фізичній локації, для забезпечення додаткового рівня захисту даних. Рекомендовано використовувати мережеві сховища (NAS) або хмарні сховища з шифруванням даних.
6. Моніторинг та управління:
    - Система моніторингу: для забезпечення безперебійної роботи інформаційної системи та своєчасного виявлення потенційних проблем необхідно реалізувати систему моніторингу. Рекомендовано використовувати інструменти моніторингу, такі як Nagios або Zabbix, з можливістю налаштування сповіщень та автоматичних дій у разі виникнення критичних подій.
    - Системи управління конфігураціями: для спрощення розгортання та управління серверною інфраструктурою рекомендовано використовувати системи управління конфігураціями, такі як Ansible або Puppet. Це дозволяє автоматизувати процеси налаштування серверів, встановлення програмного забезпечення та оновлення конфігурацій[25].
7. Безпека та захист даних:
    - Шифрування даних: для захисту конфіденційних даних, таких як паролі користувачів та чутлива інформація, необхідно використовувати шифрування. Рекомендовано використовувати

алгоритми шифрування, такі як AES або RSA, з достатньою довжиною ключів.

- Мережева безпека: для запобігання несанкціонованому доступу до мережі та серверів необхідно реалізувати заходи мережевої безпеки, такі як використання міжмережевих екранів, налаштування списків контролю доступу (ACL) та використання протоколів безпечного віддаленого доступу (наприклад, SSH).
- Автентифікація та авторизація: для забезпечення доступу до системи лише авторизованим користувачам необхідно реалізувати механізми автентифікації та авторизації. Рекомендовано використовувати багатофакторну автентифікацію (наприклад, з використанням додаткового коду підтвердження) та розмежування прав доступу на основі ролей користувачів.

Визначення технічного забезпечення також включає планування масштабованості та забезпечення можливості розширення системи в майбутньому. Необхідно враховувати потенційне збільшення кількості користувачів, обсягів даних та навантаження на систему. Тому, при виборі апаратних компонентів та планування інфраструктури, необхідно передбачити можливість вертикального та горизонтального масштабування.

Крім того, важливо забезпечити сумісність технічного забезпечення з існуючими системами та інфраструктурою організації. Необхідно врахувати інтеграцію з іншими системами, такими як системи управління проектами, системи відстеження помилок або системи керування версіями коду.

Таким чином, визначення технічного забезпечення для інформаційної системи управління тестуванням та виявленням дефектів програмного забезпечення включає вибір відповідних апаратних та програмних компонентів, планування інфраструктури та мережевого оточення, забезпечення безпеки та захисту даних, а також врахування вимог до масштабованості та сумісності. Правильно підібране технічне забезпечення є основою для ефективного функціонування та надійної роботи інформаційної системи.

3.5 Реалізація інформаційної системи

Реалізація інформаційної системи управління тестуванням та виявленням дефектів програмного забезпечення включає в себе процес розробки, конфігурування та розгортання системи відповідно до визначених вимог та проєктних рішень.

Першим кроком у реалізації системи є налаштування серверного оточення. Це включає встановлення та конфігурування серверів додатків та бази даних відповідно до визначених технічних вимог. На серверах встановлюються необхідні операційні системи, веб-сервери, системи управління базами даних та інше необхідне програмне забезпечення.

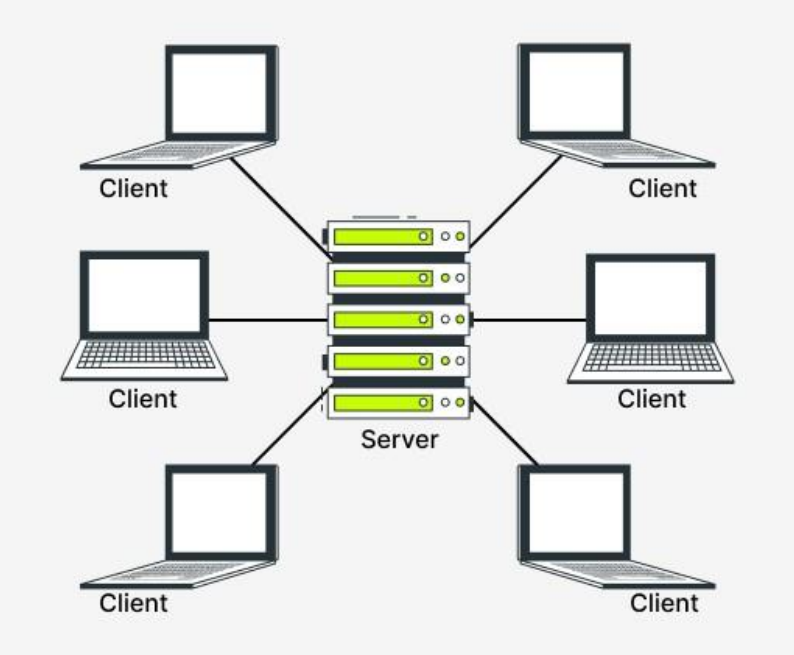

Рисунок 3.2 - Схема розгортання серверної інфраструктури

Далі відбувається розгортання веб-додатку інформаційної системи на сервері додатків. Код додатку, розроблений з використанням мови програмування PHP, розміщується на сервері та налаштовується для роботи з веб-сервером. Також проводиться конфігурація додатку, встановлення необхідних залежностей та бібліотек.

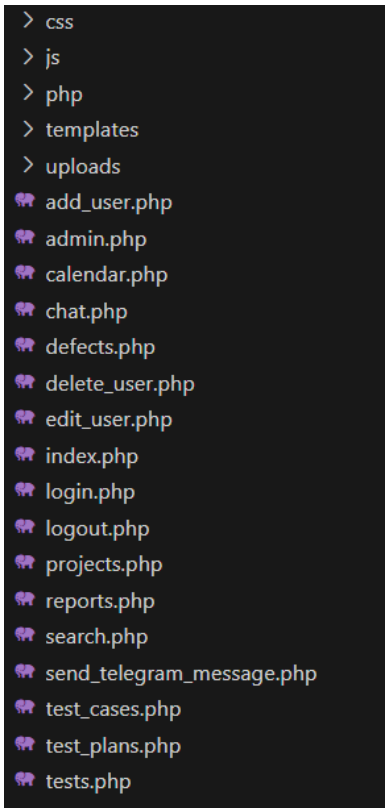

Рисунок 3.3 - Структура файлів та директорій веб-додатку

Наступним кроком є налаштування бази даних. На сервері бази даних створюється нова база даних для інформаційної системи та імпортується структура таблиць, розроблена на етапі проєктування[15]. Проводиться конфігурація підключення веб-додатку до бази даних, встановлюються відповідні драйвери та налаштовуються параметри з'єднання[16].

Після налаштування серверної частини та бази даних, проводиться тестування розгорнутої системи. Перевіряється коректність роботи всіх модулів та функцій системи, взаємодія між компонентами та відповідність результатів очікуваним. Проводяться тести на різних рівнях, включаючи модульне тестування, інтеграційне тестування та системне тестування.

Таблиця 3.1 - Результати тестування модулів системи

№	**Назва тесту**	**Очікуваний результат**	**Фактичний результат**	**Статус**
1	Авторизація користувача	Успішний вхід в систему з дійсними обліковими даними	Вхід в систему виконано успішно	Пройдено
2	Створення нового проєкту	Новий проєкт успішно створений і відображається в списку	Новий проєкт створено і відображено коректно	Пройдено

Продовження таблиці 3.1

3	Редагування існуючого проєкту	Зміни в проєкті успішно збережені і відображаються	Зміни в проєкті збережено і відображено коректно	Пройдено
4	Видалення проєкту	Проєкт успішно видалений і не відображається в списку	Проєкт видалено і він відсутній в списку	Пройдено
5	Створення нового тестового сценарію	Новий тестовий сценарій успішно створений і відображається	Новий тестовий сценарій створено і відображено коректно	Пройдено
6	Редагування існуючого тестового сценарію	Зміни в тестовому сценарії успішно збережені і відображаються	Зміни в тестовому сценарії збережено і відображено коректно	Пройдено
7	Видалення тестового сценарію	Тестовий сценарій успішно видалений і не відображається	Тестовий сценарій видалено і він відсутній в списку	Пройдено
8	Створення нового дефекту	Новий дефект успішно створений і відображається	Новий дефект створено і відображено коректно	Пройдено
9	Редагування існуючого дефекту	Зміни в дефекті успішно збережені і відображаються	Зміни в дефекті збережено і відображено коректно	Пройдено
10	Видалення дефекту	Дефект успішно видалений і не відображається	Дефект видалено і він відсутній в списку	Пройдено
11	Генерація звіту про виконання тестових планів	Звіт успішно згенерований з коректними даними	Звіт згенеровано з коректними даними	Пройдено
12	Генерація звіту про дефекти	Звіт успішно згенерований з коректними даними	Звіт згенеровано з коректними даними	Пройдено
13	Відображення загальної статистики на дашборді	Статистичні дані відображаються коректно	Статистичні дані відображено коректно	Пройдено
14	Фільтрація і сортування списку проєктів	Список проєктів коректно фільтрується і сортується	Фільтрація і сортування списку проєктів працює коректно	Пройдено
15	Призначення дефекту відповідальному користувачу	Дефект успішно призначений відповідальному користувачу	Дефект призначено відповідальному користувачу коректно	Пройдено

Після успішного тестування, відбувається наповнення системи початковими даними. Це включає створення облікових записів користувачів з відповідними правами доступу, додавання тестових проєктів, сценаріїв та планів, а також внесення іншої необхідної інформації для початкової конфігурації системи.

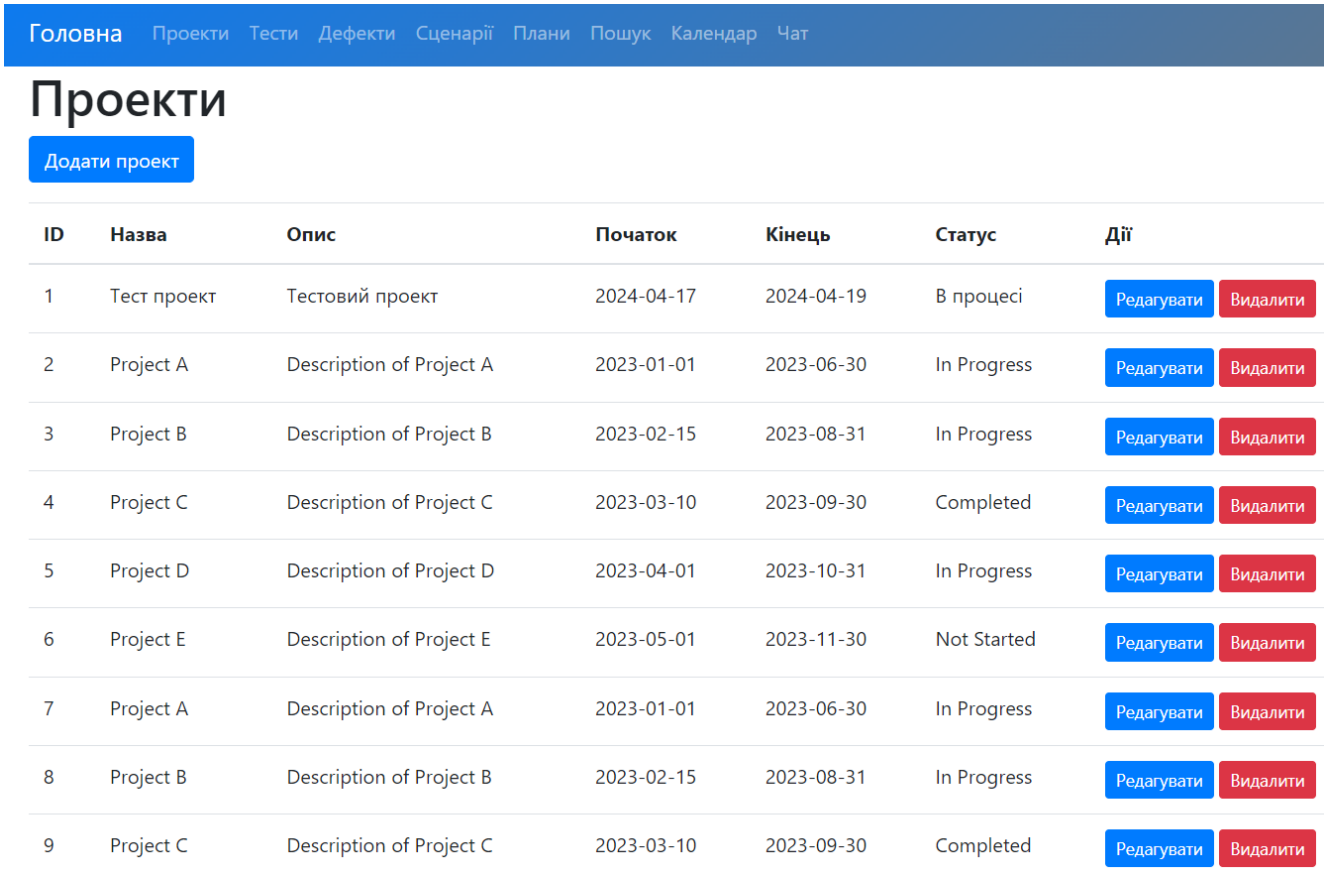

Рисунок 3.4 - Приклад наповнення системи початковими даними

Для забезпечення безпеки системи, проводиться налаштування механізмів автентифікації та авторизації користувачів. Встановлюються політики доступу, налаштовуються правила фільтрації трафіку на міжмережевому екрані та проводиться аудит безпеки системи для виявлення потенційних вразливостей.

Після завершення всіх налаштувань та тестування, інформаційна система розгортається в робочому середовищі. Користувачам надається доступ до системи через веб-інтерфейс, де вони можуть виконувати відповідні дії згідно зі своїми ролями та правами доступу.

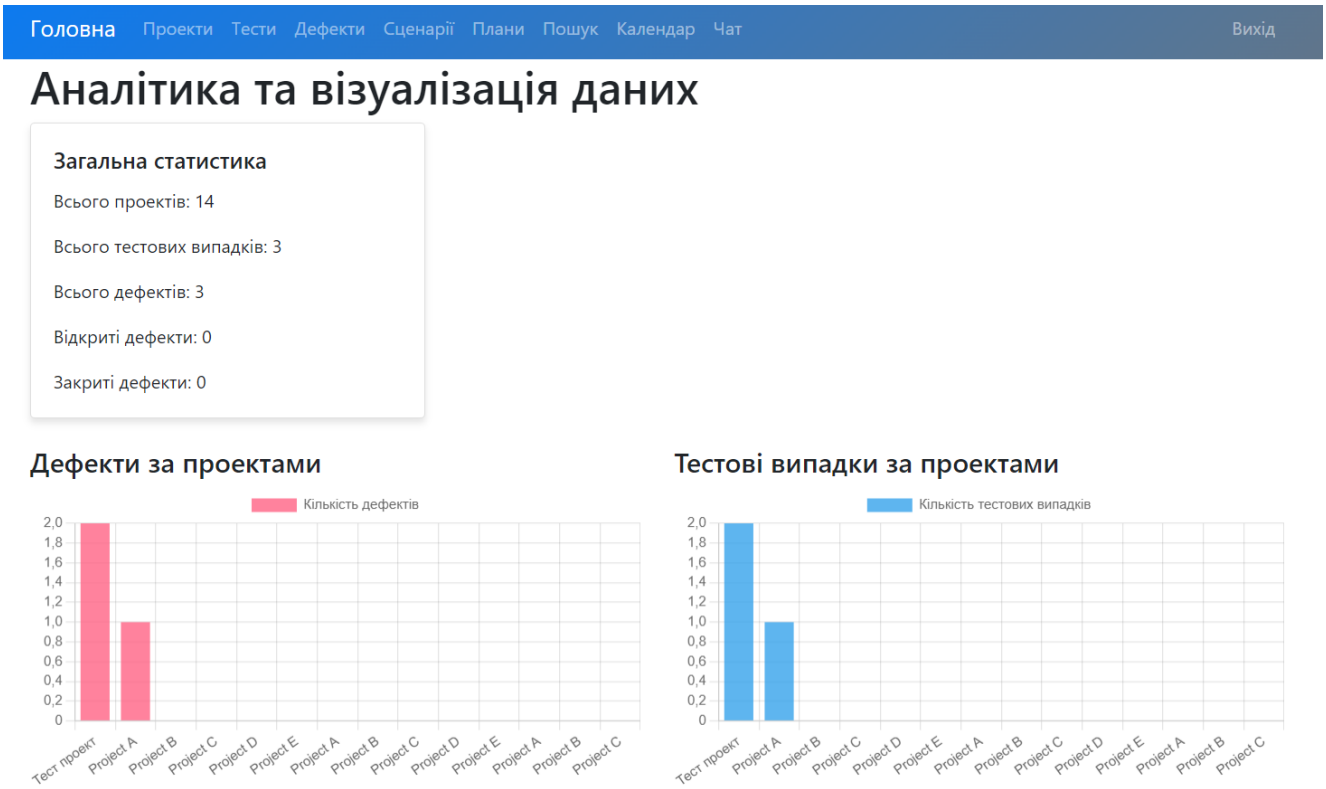

Рисунок 3.5 - Веб-інтерфейс інформаційної системи

Для забезпечення ефективної роботи користувачів з системою, проводиться навчання та підготовка персоналу. Розробляються інструкції та керівництва користувача, проводяться тренінги та семінари для ознайомлення користувачів з функціональними можливостями системи та правилами її використання.

Реалізація інформаційної системи також включає налаштування процесів резервного копіювання та відновлення даних. Розробляється розклад регулярного створення резервних копій бази даних та файлів системи, налаштовуються інструменти резервного копіювання та визначаються процедури відновлення даних у випадку збоїв або втрати інформації.

Після повної реалізації та розгортання системи, проводиться моніторинг її роботи та збір відгуків від користувачів. Відстежуються показники продуктивності, виявляються потенційні проблеми та вносяться необхідні виправлення та вдосконалення. Також проводиться регулярне оновлення системи, встановлення патчів безпеки та додавання нових функціональних можливостей відповідно до потреб користувачів та розвитку процесів тестування.

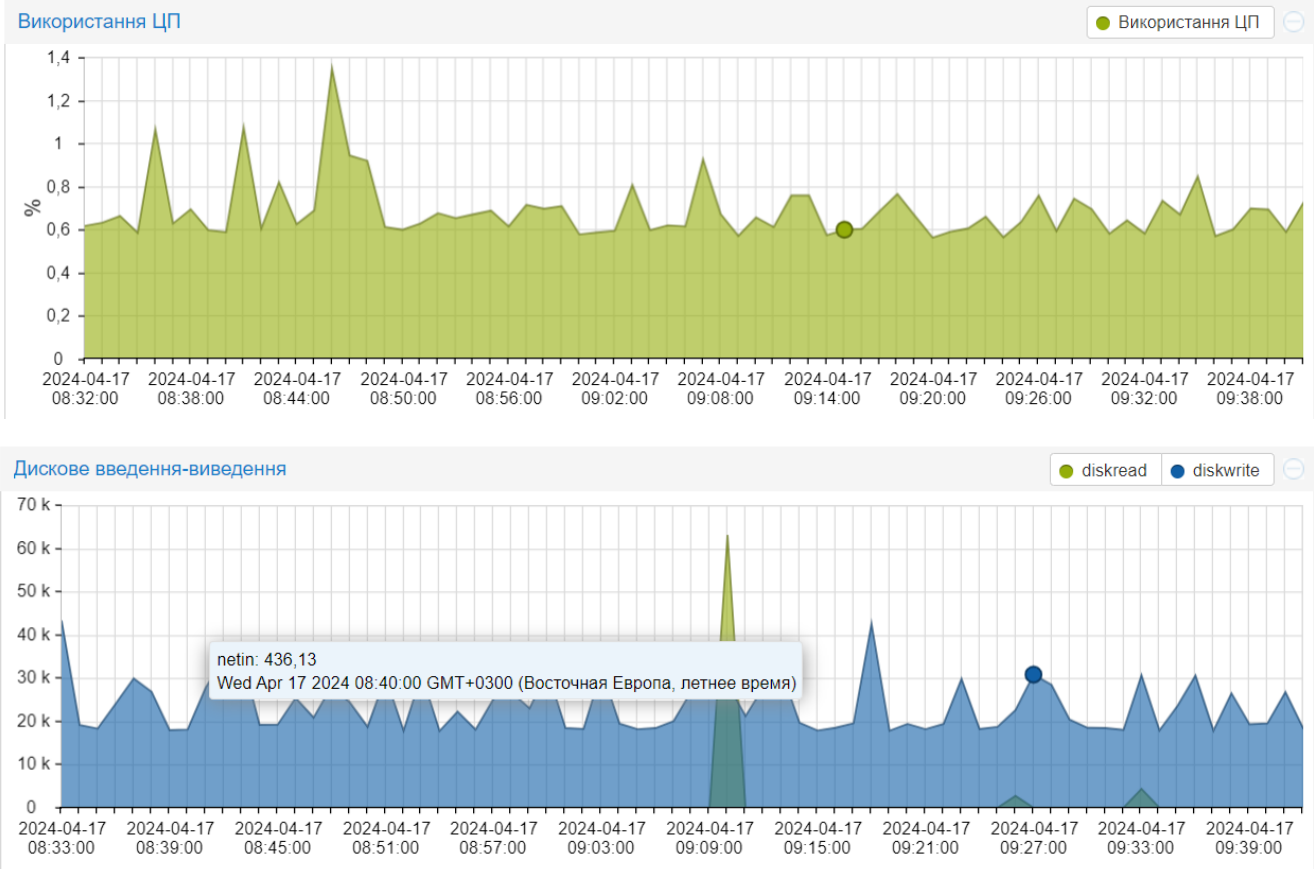

Рисунок 3.6 - Графіки моніторингу навантаження системи

Реалізація інформаційної системи управління тестуванням та виявленням дефектів програмного забезпечення є комплексним процесом, який вимагає ретельного планування, розробки, налаштування та тестування. Правильна реалізація системи забезпечує її ефективне функціонування, безпеку даних, зручність використання для користувачів та підтримку процесів тестування та управління якістю програмного забезпечення в організації.

Таким чином, реалізація інформаційної системи включає налаштування серверної інфраструктури, розгортання веб-додатку, конфігурацію бази даних, тестування системи, наповнення початковими даними, налаштування безпеки, розгортання системи в робочому середовищі, навчання користувачів та забезпечення процесів резервного копіювання та моніторингу. Успішна реалізація системи дозволяє ефективно управляти процесами тестування та виявлення дефектів, підвищувати якість програмного забезпечення та оптимізувати роботу команди тестування.

ВИСНОВКИ

У ході виконання кваліфікаційної магістерської роботи було досліджено та проаналізовано інформаційні системи управління процесами тестування та виявлення дефектів програмного забезпечення. Розглянуто існуючі рішення у цій предметній області, їх переваги та недоліки, а також сучасні підходи і технології для створення таких систем.

Проведено детальний аналіз структури та характеристик інформаційної системи у предметній області управління процесами тестування та виявлення дефектів програмного забезпечення. Розглянуто методи та моделі, які використовуються в цих системах для забезпечення ефективного управління процесами тестування та виявлення дефектів.

На основі проведеного дослідження та аналізу, було розроблено проєктні рішення для інформаційної системи управління тестуванням та виявленням дефектів програмного забезпечення. Спроєктовано базу даних, яка забезпечує ефективне зберігання та доступ до інформації про проєкти, тестові сценарії, дефекти та інші пов'язані дані[29].

Розроблено засоби інтелектуального аналізу даних, які дозволяють виявляти закономірності, тенденції та взаємозв'язки у даних про тестування та дефекти[39]. Ці засоби допомагають приймати обґрунтовані рішення щодо планування тестування, розподілу ресурсів та вдосконалення процесів розробки програмного забезпечення.

Реалізовано веб-додаток інформаційної системи з використанням сучасних технологій та фреймворків, таких як PHP та Laravel. Розроблено різноманітні модулі та функції, які забезпечують ефективне управління проєктами, тестовими сценаріями, дефектами, генерацію звітів та аналітику даних.

Визначено технічне забезпечення, необхідне для ефективного функціонування інформаційної системи. Розроблено вимоги до серверного обладнання, мережевої інфраструктури, клієнтських робочих станцій та програмного забезпечення. Забезпечено безпеку та захист даних шляхом використання механізмів шифрування, автентифікації та авторизації користувачів.

Реалізовано розгортання інформаційної системи в робочому середовищі, проведено тестування та наповнення системи початковими даними. Розроблено інструкції та проведено навчання користувачів для ефективної роботи з системою. Налаштовано процеси резервного копіювання та моніторингу продуктивності системи.

Таким чином, розроблена інформаційна система управління тестуванням та виявленням дефектів програмного забезпечення дозволяє ефективно управляти процесами тестування, відстежувати дефекти, генерувати звіти та проводити аналітику даних. Система забезпечує підвищення якості програмного забезпечення, оптимізацію роботи команди тестування та прийняття обґрунтованих рішень на основі даних.

Результати кваліфікаційної магістерської роботи можуть бути використані для подальшого вдосконалення процесів тестування та управління якістю програмного забезпечення в організаціях. Розроблена інформаційна система може слугувати основою для майбутніх досліджень та розширення функціональних можливостей у цій предметній області.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Морозов, А. В. Інформаційні системи управління проектами: навчальний посібник / А. В. Морозов, О. В. Косенко, Л. О. Отрода. – Харків: ХНУМГ ім. О. М. Бекетова, 2019. – 127 с.
2. Куправа, Т. А. Управління проектами: навчальний посібник / Т. А. Куправа. – К.: КНЕУ, 2017. – 364 с.
3. Бабенко, Л. П. Основи програмної інженерії: навчальний посібник / Л. П. Бабенко, К. М. Лавріщева. – К.: Знання, 2001. – 269 с.
4. Sommerville, I. Software Engineering / I. Sommerville. – 10th ed. – Pearson, 2015. – 816 p.
5. Синицын, С. В. Верификация программного обеспечения: учебное пособие / С. В. Синицын, Н. Ю. Налютин. – М.: БИНОМ, 2008. – 368 с.
6. Spillner, A. Software Testing Foundations: A Study Guide for the Certified Tester Exam / A. Spillner, T. Linz, H. Schaefer. – 4th ed. – Rocky Nook, 2014. – 536 p.
7. Буч, Г. Введение в UML от создателей языка / Г. Буч, Д. Рамбо, И. Якобсон. – 2-е изд. – М.: ДМК Пресс, 2015. – 496 с.
8. Fowler, M. UML Distilled: A Brief Guide to the Standard Object Modeling Language / M. Fowler. – 3rd ed. – Addison-Wesley Professional, 2003. – 208 p.
9. Гради Буч. Объектно-ориентированный анализ и проектирование с примерами приложений / Гради Буч, Роберт А. Максимчук, Майкл У. Энгл и др. – 3-е изд. – М.: Вильямс, 2010. – 720 с.
10. Larman, C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development / C. Larman. – 3rd ed. – Prentice Hall, 2004. – 736 p.
11. Голдштейн, С. Справочник по технологиям и средствам связи / С. Голдштейн, І. Ехріель, Р. Рерле. – М.: Радио и связь, 2005. – 488 с.
12. Forouzan, B. A. Data Communications and Networking / B. A. Forouzan. – 5th ed. – McGraw-Hill, 2012. – 1264 p.
13. Дейтел, П. Как программировать на PHP / П. Дейтел, Х. Дейтел, Д. Макпик. – 5-е изд. – М.: Бином, 2009. – 1104 с.

14. Sklar, D. Learning PHP: A Gentle Introduction to the Web's Most Popular Language / D. Sklar. – O'Reilly Media, 2016. – 408 p.
15. Лакетт, Д. Администрирование PostgreSQL 9 / Д. Лакетт, А. Валиков. – М.: ДМК Пресс, 2013. – 368 с.
16. Obe, R. O. PostgreSQL: Up and Running: A Practical Guide to the Advanced Open Source Database / R. O. Obe, L. S. Hsu. – 3rd ed. – O'Reilly Media, 2017. – 342 p.
17. Шаров, С. В. Мова програмування Java: навчальний посібник / С. В. Шаров, В. І. Сердюк. – Мелітополь: Видавництво МДПУ ім. Б. Хмельницького, 2016. – 184 с.
18. Bloch, J. Effective Java / J. Bloch. – 3rd ed. – Addison-Wesley Professional, 2017. – 412 p.
19. Гамма, Э. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Д. Влиссидес. – СПб.: Питер, 2015. – 368 с.
20. Freeman, E. Head First Design Patterns: A Brain-Friendly Guide / E. Freeman, E. Robson, B. Bates, K. Sierra. – O'Reilly Media, 2004. – 694 p.
21. Ткаченко, О. М. Методи та засоби тестування програмного забезпечення: навчальний посібник / О. М. Ткаченко, Н. В. Загородна, Т. П. Зубрецька. – Вінниця: ВНТУ, 2015. – 80 с.
22. Myers, G. J. The Art of Software Testing / G. J. Myers, C. Sandler, T. Badgett. – 3rd ed. – Wiley, 2011. – 256 p.
23. Шилдт, Г. Java. Полное руководство / Г. Шилдт. – 10-е изд. – М.: Диалектика, 2018. – 1488 с.
24. Horstmann, C. Core Java Volume I--Fundamentals / C. Horstmann. – 11th ed. – Prentice Hall, 2018. – 928 p.
25. Захарченко, В. В. Клієнт-серверні технології: навчальний посібник / В. В. Захарченко. – Одеса: ОНАЗ ім. О.С. Попова, 2018. – 88 с.
26. Tanenbaum, A. S. Distributed Systems: Principles and Paradigms / A. S. Tanenbaum, M. van Steen. – 2nd ed. – Pearson, 2006. – 686 p.

27. Олещук, А. В. Розробка веб-додатків з використанням мови програмування PHP: навчальний посібник / А. В. Олещук. – К.: КНУБА, 2019. – 196 с.
28. Welling, L. PHP and MySQL Web Development / L. Welling, L. Thomson. – 5th ed. – Addison-Wesley Professional, 2016. – 688 p.
29. Шевченко, Н. А. Основи проектування реляційних баз даних: навчальний посібник / Н. А. Шевченко. – К.: КНУБА, 2020. – 126 с.
30. Elmasri, R. Fundamentals of Database Systems / R. Elmasri, S. B. Navathe. – 7th ed. – Pearson, 2015. – 1272 p.
31. Кузнецов, М. В. MySQL 5 / М. В. Кузнецов, И. В. Симдянов. – СПб.: БХВ-Петербург, 2010. – 1024 с.
32. Schwartz, B. High Performance MySQL: Optimization, Backups, and Replication / B. Schwartz, P. Zaitsev, V. Tkachenko. – 3rd ed. – O'Reilly Media, 2012. – 828 p.
33. Мартин, Р. Чистий код / Р. Мартин. – К.: Фабула, 2019. – 448 с.
34. Martin, R. C. Clean Code: A Handbook of Agile Software Craftsmanship / R. C. Martin. – Prentice Hall, 2008. – 464 p.
35. Чаплига, В. М. Тестування програмного забезпечення: навчальний посібник / В. М. Чаплига, П. М. Чаплига. – Львів: Львівська політехніка, 2016. – 136 с.
36. Humble, J. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation / J. Humble, D. Farley. – Addison-Wesley Professional, 2010. – 512 p.
37. Бейзер, Б. Тестирование черного ящика / Б. Бейзер. – СПб.: Питер, 2004. – 320 с.
38. Copeland, L. A Practitioner's Guide to Software Test Design / L. Copeland. – Artech House, 2003. – 304 p.
39. Козловський, А. В. Інтелектуальний аналіз даних: навчальний посібник / А. В. Козловський, Н. І. Поворознюк, В. П. Козловський. – Львів: Видавництво Львівської політехніки, 2018. – 260 с.
40. Han, J. Data Mining: Concepts and Techniques / J. Han, M. Kamber, J. Pei. – 3rd ed. – Morgan Kaufmann, 2011. – 744 p.

ДОДАТКИ

Код файлу functions.php.

```
<?php
require_once 'database.php';

if (session_status() == PHP_SESSION_NONE) {
    session_start();
}

function isLoggedIn() {
    return isset($_SESSION['user_id']);
}

function authenticateUser($username, $password) {
    global $db;
    $query = "SELECT * FROM users WHERE username = :username";
    $stmt = $db->prepare($query);
    $stmt->execute(['username' => $username]);
    $user = $stmt->fetch(PDO::FETCH_ASSOC);

    if ($user && password_verify($password, $user['password'])) {
        $_SESSION['user_id'] = $user['id'];
        $_SESSION['username'] = $user['username'];
        $_SESSION['user'] = $user;
        return true;
    }

    return false;
}
```

```
function userExists($username) {
    global $db;
    $query = "SELECT * FROM users WHERE username = :username";
    $stmt = $db->prepare($query);
    $stmt->execute(['username' => $username]);
    $user = $stmt->fetch(PDO::FETCH_ASSOC);
    return $user ? true : false;
}

function registerUser($username, $password) {
    global $db;
      $query = "INSERT INTO users (username, password) VALUES (:username, :password)";
    $stmt = $db->prepare($query);
    $result = $stmt->execute([
        'username' => $username,
        'password' => password_hash($password, PASSWORD_DEFAULT),
    ]);

    return $result;
}

function logoutUser() {
    unset($_SESSION['user_id']);
     unset($_SESSION['username']);
     unset($_SESSION['user']);
 }

 function getUsers()
```

```
{
    global $db;

    $query = "SELECT * FROM users";
    $stmt = $db->prepare($query);
    $stmt->execute();

    return $stmt->fetchAll(PDO::FETCH_ASSOC);
}
function addUser($username, $password, $hourly_rate)
{
    global $db;

    $hashed_password = password_hash($password, PASSWORD_DEFAULT);
      $query = "INSERT INTO users (username, password, hourly_rate) VALUES (:username, :password, :hourly_rate)";
    $stmt = $db->prepare($query);
      $stmt->execute([':username' => $username, ':password' => $hashed_password, ':hourly_rate' => $hourly_rate]);
}

function getUserById($id)
{
    global $db;

    $query = "SELECT * FROM users WHERE id = :id";
    $stmt = $db->prepare($query);
    $stmt->execute([':id' => $id]);

    return $stmt->fetch(PDO::FETCH_ASSOC);
}
```

```
function updateUser($id, $new_id, $username, $password, $hourly_rate)
{
    global $db;

    $hashed_password = password_hash($password, PASSWORD_DEFAULT);
     $query = "UPDATE users SET id = :new_id, username = :username, password = :password, hourly_rate = :hourly_rate WHERE id = :id";
    $stmt = $db->prepare($query);
     $stmt->execute([':new_id' => $new_id, ':username' => $username, ':password' => $hashed_password, ':id' => $id, ':hourly_rate' => $hourly_rate]);
}

function deleteUser($id)
{
    global $db;

    $query = "DELETE FROM users WHERE id = :id";
    $stmt = $db->prepare($query);
    $stmt->execute([':id' => $id]);
}

/* Chat*/

function getChatMessages($userId = null) {
    global $db;
     $query = "SELECT chat_messages.*, users.username FROM chat_messages JOIN users ON chat_messages.user_id = users.id ORDER BY chat_messages.timestamp";
    $stmt = $db->prepare($query);
    $stmt->execute();
    $messages = $stmt->fetchAll(PDO::FETCH_ASSOC);
```

```
    // присвоюємо кожноу повідомлення ім’я користувача, якщо існує
    foreach ($messages as &$msg) {
        if (!empty($msg['username'])) {
            $msg['user'] = [
                'id' => $msg['user_id'],
                'username' => $msg['username']
            ];
        }
        unset($msg['username']);
    }

    if ($userId) {
        $messages = array_filter($messages, function ($msg) use ($userId) {
            return $msg['user']['id'] === $userId;
        });
    }

    return $messages;
}

function addChatMessage($user_id, $message, $task_time) {
    global $db;
     $query = "INSERT INTO chat_messages (user_id, message, task_time) VALUES
(:user_id, :message, :task_time)";
    $stmt = $db->prepare($query);
    $result = $stmt->execute([
        'user_id' => $user_id,
        'message' => $message,
        'task_time' => $task_time,
    ]);
```

```
    return $result;
}

function deleteChatMessage($id) {
    global $db;
    $query = "DELETE FROM chat_messages WHERE id = :id";
    $stmt = $db->prepare($query);
    $result = $stmt->execute(['id' => $id]);

    return $result;
}

function updateChatMessage($id, $message, $task_time) {
    global $db;
    $query = "UPDATE chat_messages SET message = :message, task_time = :task_time
WHERE id = :id";
    $stmt = $db->prepare($query);
    $result = $stmt->execute([
        'id' => $id,
        'message' => $message,
        'task_time' => $task_time,
    ]);

    return $result;
}

function getWorkers() {
    global $db;

    $query = "SELECT id, full_name, position, employment_date FROM workers";
```

```
    $stmt = $db->prepare($query);
    $stmt->execute();

    return $stmt->fetchAll(PDO::FETCH_ASSOC);
}

//проєкти
function getProjects() {
    global $db;
    $query = "SELECT * FROM projects";
    $stmt = $db->prepare($query);
    $stmt->execute();
    return $stmt->fetchAll(PDO::FETCH_ASSOC);
}

function getProjectById($id) {
    global $db;
    $query = "SELECT * FROM projects WHERE id = :id";
    $stmt = $db->prepare($query);
    $stmt->execute(['id' => $id]);
    return $stmt->fetch(PDO::FETCH_ASSOC);
}

function addProject($name, $description, $start_date, $end_date, $status) {
    global $db;
     $query = "INSERT INTO projects (name, description, start_date, end_date, status)
VALUES (:name, :description, :start_date, :end_date, :status)";
    $stmt = $db->prepare($query);
    $stmt->execute([
        'name' => $name,
        'description' => $description,
```

```
        'start_date' => $start_date,
        'end_date' => $end_date,
        'status' => $status,
    ]);
    return $db->lastInsertId();
}

function updateProject($id, $name, $description, $start_date, $end_date, $status) {
    global $db;
    $query = "UPDATE projects SET name = :name, description = :description, start_date
= :start_date, end_date = :end_date, status = :status WHERE id = :id";
    $stmt = $db->prepare($query);
    $stmt->execute([
        'id' => $id,
        'name' => $name,
        'description' => $description,
        'start_date' => $start_date,
        'end_date' => $end_date,
        'status' => $status,
    ]);
}

function deleteProject($id) {
    global $db;
    $query = "DELETE FROM projects WHERE id = :id";
    $stmt = $db->prepare($query);
    $stmt->execute(['id' => $id]);
}
```

//mecmu

```
function getTests() {
    global $db;
     $query = "SELECT tests.*, projects.name AS project_name, users.username AS user_name FROM tests
            JOIN projects ON tests.project_id = projects.id
            JOIN users ON tests.user_id = users.id";
    $stmt = $db->prepare($query);
    $stmt->execute();
    return $stmt->fetchAll(PDO::FETCH_ASSOC);
}

function getTestById($id) {
    global $db;
     $query = "SELECT tests.*, projects.name AS project_name, users.username AS user_name FROM tests
            JOIN projects ON tests.project_id = projects.id
            JOIN users ON tests.user_id = users.id
            WHERE tests.id = :id";
    $stmt = $db->prepare($query);
    $stmt->execute(['id' => $id]);
    return $stmt->fetch(PDO::FETCH_ASSOC);
}

function addTest($project_id, $user_id, $test_type, $test_result, $version, $description) {
    global $db;
     $query = "INSERT INTO tests (project_id, user_id, test_type, test_result, version, description)
            VALUES (:project_id, :user_id, :test_type, :test_result, :version, :description)";
    $stmt = $db->prepare($query);
    $stmt->execute([
```

```
        'project_id' => $project_id,
        'user_id' => $user_id,
        'test_type' => $test_type,
        'test_result' => $test_result,
        'version' => $version,
        'description' => $description,
    ]);
    return $db->lastInsertId();
}

function updateTest($id, $project_id, $user_id, $test_type, $test_result, $version, $description) {
    global $db;
    $query = "UPDATE tests SET project_id = :project_id, user_id = :user_id, test_type = :test_type,
              test_result = :test_result, version = :version, description = :description
              WHERE id = :id";
    $stmt = $db->prepare($query);
    $stmt->execute([
        'id' => $id,
        'project_id' => $project_id,
        'user_id' => $user_id,
        'test_type' => $test_type,
        'test_result' => $test_result,
        'version' => $version,
        'description' => $description,
    ]);
}

function deleteTest($id) {
    global $db;
```

```
    $query = "DELETE FROM tests WHERE id = :id";
    $stmt = $db->prepare($query);
    $stmt->execute(['id' => $id]);
}

//багтрекер

function getDefects($projectId = null) {
    global $db;
     $query = "SELECT defects.*, projects.name AS project_name, users.username AS assigned_to_username
              FROM defects
              JOIN projects ON defects.project_id = projects.id
              LEFT JOIN users ON defects.assigned_to = users.id";
    if ($projectId) {
        $query .= " WHERE defects.project_id = :project_id";
    }
    $stmt = $db->prepare($query);
    if ($projectId) {
        $stmt->execute(['project_id' => $projectId]);
    } else {
        $stmt->execute();
    }
    return $stmt->fetchAll(PDO::FETCH_ASSOC);
}

function getDefectById($id) {
    global $db;
     $query = "SELECT defects.*, projects.name AS project_name, users.username AS assigned_to_username
              FROM defects
```

```
        JOIN projects ON defects.project_id = projects.id
        LEFT JOIN users ON defects.assigned_to = users.id
        WHERE defects.id = :id";
    $stmt = $db->prepare($query);
    $stmt->execute(['id' => $id]);
    return $stmt->fetch(PDO::FETCH_ASSOC);
}

function addDefect($projectId, $title, $description, $status, $assignedTo) {
    global $db;
    $query = "INSERT INTO defects (project_id, title, description, status, assigned_to)
        VALUES (:project_id, :title, :description, :status, :assigned_to)";
    $stmt = $db->prepare($query);
    $stmt->execute([
        'project_id' => $projectId,
        'title' => $title,
        'description' => $description,
        'status' => $status,
        'assigned_to' => $assignedTo
    ]);
    return $db->lastInsertId();
}

function updateDefect($id, $projectId, $title, $description, $status, $assignedTo) {
    global $db;
    $query = "UPDATE defects SET project_id = :project_id, title = :title, description =
:description,
        status = :status, assigned_to = :assigned_to WHERE id = :id";
    $stmt = $db->prepare($query);
    $stmt->execute([
        'id' => $id,
```

```
        'project_id' => $projectId,
        'title' => $title,
        'description' => $description,
        'status' => $status,
        'assigned_to' => $assignedTo
    ]);
}

function deleteDefect($id) {
    global $db;
    $query = "DELETE FROM defects WHERE id = :id";
    $stmt = $db->prepare($query);
    $stmt->execute(['id' => $id]);
}

//Сценарії
function getTestCases($projectId = null) {
    global $db;
    $query = "SELECT * FROM test_cases";
    if ($projectId) {
        $query .= " WHERE project_id = :project_id";
    }
    $stmt = $db->prepare($query);
    if ($projectId) {
        $stmt->execute(['project_id' => $projectId]);
    } else {
        $stmt->execute();
    }
    return $stmt->fetchAll(PDO::FETCH_ASSOC);
}
```

```
function getTestCaseById($id) {
    global $db;
    $query = "SELECT * FROM test_cases WHERE id = :id";
    $stmt = $db->prepare($query);
    $stmt->execute(['id' => $id]);
    return $stmt->fetch(PDO::FETCH_ASSOC);
}

function addTestCase($title, $description, $projectId, $module) {
    global $db;
    $query = "INSERT INTO test_cases (title, description, project_id, module)
            VALUES (:title, :description, :project_id, :module)";
    $stmt = $db->prepare($query);
    $stmt->execute([
        'title' => $title,
        'description' => $description,
        'project_id' => $projectId,
        'module' => $module
    ]);
    return $db->lastInsertId();
}

function updateTestCase($id, $title, $description, $projectId, $module) {
    global $db;
    $query = "UPDATE test_cases SET title = :title, description = :description,
            project_id = :project_id, module = :module WHERE id = :id";
    $stmt = $db->prepare($query);
    $stmt->execute([
        'id' => $id,
        'title' => $title,
        'description' => $description,
```

```
        'project_id' => $projectId,
        'module' => $module
    ]);
}

function deleteTestCase($id) {
    global $db;
    $query = "DELETE FROM test_cases WHERE id = :id";
    $stmt = $db->prepare($query);
    $stmt->execute(['id' => $id]);
}

function importTestCases($projectId, $csvData) {
    global $db;
    $lines = explode("\n", $csvData);
    $query = "INSERT INTO test_cases (title, description, project_id, module) VALUES
(?, ?, ?, ?)";
    $stmt = $db->prepare($query);

    foreach ($lines as $line) {
        if (empty($line)) {
            continue;
        }
        $data = str_getcsv($line);
        $stmt->execute($data);
    }
}

function exportTestCases($projectId) {
    $testCases = getTestCases($projectId);
    $csvData = "Title,Description,Module\n";
```

```
    foreach ($testCases as $testCase) {
                                                    $csvData .=
"{$testCase['title']},{$testCase['description']},{$testCase['module']}\n";
    }

    return $csvData;
}

//плани
function getTestPlans($projectId = null) {
    global $db;
    $query = "SELECT * FROM test_plans";
    if ($projectId) {
        $query .= " WHERE project_id = :project_id";
    }
    $stmt = $db->prepare($query);
    if ($projectId) {
        $stmt->execute(['project_id' => $projectId]);
    } else {
        $stmt->execute();
    }
    return $stmt->fetchAll(PDO::FETCH_ASSOC);
}

function getTestPlanById($id) {
    global $db;
    $query = "SELECT * FROM test_plans WHERE id = :id";
    $stmt = $db->prepare($query);
    $stmt->execute(['id' => $id]);
    return $stmt->fetch(PDO::FETCH_ASSOC);
```

```
}

function addTestPlan($projectId, $name, $description, $startDate, $endDate, $status) {
    global $db;
     $query = "INSERT INTO test_plans (project_id, name, description, start_date, end_date, status)
             VALUES (:project_id, :name, :description, :start_date, :end_date, :status)";
    $stmt = $db->prepare($query);
    $stmt->execute([
        'project_id' => $projectId,
        'name' => $name,
        'description' => $description,
        'start_date' => $startDate,
        'end_date' => $endDate,
        'status' => $status
    ]);
    return $db->lastInsertId();
}

function updateTestPlan($id, $projectId, $name, $description, $startDate, $endDate, $status) {
    global $db;
     $query = "UPDATE test_plans SET project_id = :project_id, name = :name, description = :description,
             start_date = :start_date, end_date = :end_date, status = :status WHERE id = :id";
    $stmt = $db->prepare($query);
    $stmt->execute([
        'id' => $id,
        'project_id' => $projectId,
        'name' => $name,
        'description' => $description,
```

```
        'start_date' => $startDate,
        'end_date' => $endDate,
        'status' => $status
    ]);
}

function deleteTestPlan($id) {
    global $db;
    $query = "DELETE FROM test_plans WHERE id = :id";
    $stmt = $db->prepare($query);
    $stmt->execute(['id' => $id]);
}

//головна
function getTestingMetrics() {
    global $db;
    $query = "SELECT
            (SELECT COUNT(*) FROM projects) AS total_projects,
            (SELECT COUNT(*) FROM test_cases) AS total_test_cases,
            (SELECT COUNT(*) FROM defects) AS total_defects,
               (SELECT COUNT(*) FROM defects WHERE status = 'Відкритий') AS open_defects,
                (SELECT COUNT(*) FROM defects WHERE status = 'Закритий') AS closed_defects
            FROM dual";
    $stmt = $db->prepare($query);
    $stmt->execute();
    return $stmt->fetch(PDO::FETCH_ASSOC);
}

function getDefectsByProject() {
```

```
    global $db;
      $query = "SELECT projects.name AS project_name, COUNT(defects.id) AS defect_count
            FROM projects
            LEFT JOIN defects ON projects.id = defects.project_id
            GROUP BY projects.id";
    $stmt = $db->prepare($query);
    $stmt->execute();
    return $stmt->fetchAll(PDO::FETCH_ASSOC);
}

function getTestCasesByProject() {
    global $db;
     $query = "SELECT projects.name AS project_name, COUNT(test_cases.id) AS test_case_count
            FROM projects
            LEFT JOIN test_cases ON projects.id = test_cases.project_id
            GROUP BY projects.id";
    $stmt = $db->prepare($query);
    $stmt->execute();
    return $stmt->fetchAll(PDO::FETCH_ASSOC);
}

//пошук
function searchEntities($keyword) {
    global $db;
    $keyword = '%' . $keyword . '%';

    $query = "SELECT 'project' AS entity_type, id, name AS title, description
            FROM projects
            WHERE name LIKE :keyword OR description LIKE :keyword
```

```
        UNION ALL

        SELECT 'test_case' AS entity_type, id, title, description
        FROM test_cases
        WHERE title LIKE :keyword OR description LIKE :keyword

        UNION ALL

        SELECT 'defect' AS entity_type, id, title, description
        FROM defects
        WHERE title LIKE :keyword OR description LIKE :keyword";

    $stmt = $db->prepare($query);
    $stmt->bindValue(':keyword', $keyword, PDO::PARAM_STR);
    $stmt->execute();

    return $stmt->fetchAll(PDO::FETCH_ASSOC);
}

//календар
function getEvents() {
    global $db;
    $query = "SELECT * FROM events ORDER BY start_date";
    $stmt = $db->prepare($query);
    $stmt->execute();
    return $stmt->fetchAll(PDO::FETCH_ASSOC);
}

function getEventById($id) {
    global $db;
```

```
    $query = "SELECT * FROM events WHERE id = :id";
    $stmt = $db->prepare($query);
    $stmt->execute(['id' => $id]);
    return $stmt->fetch(PDO::FETCH_ASSOC);
}

function addEvent($title, $description, $startDate, $endDate) {
    global $db;
    $query = "INSERT INTO events (title, description, start_date, end_date)
            VALUES (:title, :description, :start_date, :end_date)";
    $stmt = $db->prepare($query);
    $stmt->execute([
        'title' => $title,
        'description' => $description,
        'start_date' => $startDate,
        'end_date' => $endDate
    ]);
    return $db->lastInsertId();
}

function updateEvent($id, $title, $description, $startDate, $endDate) {
    global $db;
    $query = "UPDATE events SET title = :title, description = :description,
            start_date = :start_date, end_date = :end_date WHERE id = :id";
    $stmt = $db->prepare($query);
    $stmt->execute([
        'id' => $id,
        'title' => $title,
        'description' => $description,
        'start_date' => $startDate,
        'end_date' => $endDate
```

```
    ]);
}

function deleteEvent($id) {
    global $db;
    $query = "DELETE FROM events WHERE id = :id";
    $stmt = $db->prepare($query);
    $stmt->execute(['id' => $id]);
}
```

Код з файлу projects.php

```
<?php
session_start();
require_once 'php/functions.php';

if (!isLoggedIn()) {
    header('Location: login.php');
    exit;
}

$projects = getProjects();

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    if (isset($_POST['action'])) {
        $action = $_POST['action'];

        switch ($action) {
            case 'add':
                $name = $_POST['name'];
                $description = $_POST['description'];
                $start_date = $_POST['start_date'];
                $end_date = $_POST['end_date'];
```

```
        $status = $_POST['status'];

        $projectId = addProject($name, $description, $start_date, $end_date, $status);
        echo json_encode(['success' => true, 'projectId' => $projectId]);
        break;

    case 'edit':
        $id = $_POST['id'];
        $project = getProjectById($id);
        echo json_encode($project);
        break;

    case 'update':
        $id = $_POST['id'];
        $name = $_POST['name'];
        $description = $_POST['description'];
        $start_date = $_POST['start_date'];
        $end_date = $_POST['end_date'];
        $status = $_POST['status'];

        updateProject($id, $name, $description, $start_date, $end_date, $status);
        echo json_encode(['success' => true]);
        break;

    case 'delete':
        $id = $_POST['id'];
        deleteProject($id);
        echo json_encode(['success' => true]);
        break;
}
```

```
        exit;
    }
}

include 'templates/header.php';
?>

<div class="container">
    <h1>Проекти</h1>
    <button type="button" class="btn btn-primary mb-3" data-bs-toggle="modal" data-bs-target="#addProjectModal">Додати проект</button>
    <table class="table">
        <thead>
            <tr>
                <th>ID</th>
                <th>Назва</th>
                <th>Опис</th>
                <th>Початок</th>
                <th>Кінець</th>
                <th>Статус</th>
                <th>Дії</th>
            </tr>
        </thead>
        <tbody>
            <?php foreach ($projects as $project): ?>
                <tr>
                    <td><?= $project['id'] ?></td>
                    <td><?= $project['name'] ?></td>
                    <td><?= $project['description'] ?></td>
                    <td><?= $project['start_date'] ?></td>
                    <td><?= $project['end_date'] ?></td>
```

```
                <td><?= $project['status'] ?></td>
                <td>
                    <button type="button" class="btn btn-primary btn-sm edit-project" data-id="<?= $project['id'] ?>">Редагувати</button>
                    <button type="button" class="btn btn-danger btn-sm delete-project" data-id="<?= $project['id'] ?>">Видалити</button>
                </td>
            </tr>
        <?php endforeach; ?>
    </tbody>
  </table>
</div>

<!-- Модальне вікно для додавання проекту -->
<div class="modal fade" id="addProjectModal" tabindex="-1" aria-labelledby="addProjectModalLabel" aria-hidden="true">
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title" id="addProjectModalLabel">Додати проект</h5>
            <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>
      </div>
      <div class="modal-body">
        <form id="addProjectForm">
          <div class="mb-3">
            <label for="name" class="form-label">Назва</label>
                <input type="text" class="form-control" id="name" name="name" required>
          </div>
          <div class="mb-3">
```

```
                <label for="description" class="form-label">Опис</label>
                    <textarea class="form-control" id="description" name="description" rows="3"></textarea>
                </div>
                <div class="mb-3">
                    <label for="start_date" class="form-label">Дата початку</label>
                                <input type="date" class="form-control" id="start_date" name="start_date">
                </div>
                <div class="mb-3">
                    <label for="end_date" class="form-label">Дата завершення</label>
                 <input type="date" class="form-control" id="end_date" name="end_date">
                </div>
                <div class="mb-3">
                    <label for="status" class="form-label">Статус</label>
                    <select class="form-select" id="status" name="status">
                        <option value="Новий">Новий</option>
                        <option value="В процесі">В процесі</option>
                        <option value="Завершений">Завершений</option>
                    </select>
                </div>
            </form>
        </div>
        <div class="modal-footer">
                            <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Закрити</button>
                                <button type="button" class="btn btn-primary" id="saveProject">Зберегти</button>
        </div>
    </div>
</div>
```

```
</div>

<!-- Модальне вікно для редагування проекту -->
<div class="modal fade" id="editProjectModal" tabindex="-1" aria-labelledby="editProjectModalLabel" aria-hidden="true">
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title" id="editProjectModalLabel">Редагувати проект</h5>
            <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>
      </div>
      <div class="modal-body">
        <form id="editProjectForm">
          <input type="hidden" id="editProjectId" name="id">
          <div class="mb-3">
            <label for="editName" class="form-label">Назва</label>
              <input type="text" class="form-control" id="editName" name="name" required>
          </div>
          <div class="mb-3">
            <label for="editDescription" class="form-label">Опис</label>
            <textarea class="form-control" id="editDescription" name="description" rows="3"></textarea>
          </div>
          <div class="mb-3">
            <label for="editStartDate" class="form-label">Дата початку</label>
                    <input type="date" class="form-control" id="editStartDate" name="start_date">
          </div>
          <div class="mb-3">
```

```
                <label for="editEndDate" class="form-label">Дата завершення</label>
                    <input type="date" class="form-control" id="editEndDate" name="end_date">
                </div>
                <div class="mb-3">
                    <label for="editStatus" class="form-label">Статус</label>
                    <select class="form-select" id="editStatus" name="status">
                        <option value="Новий">Новий</option>
                        <option value="В процесі">В процесі</option>
                        <option value="Завершений">Завершений</option>
                    </select>
                </div>
            </form>
        </div>
        <div class="modal-footer">
                    <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Закрити</button>
            <button type="button" class="btn btn-primary" id="updateProject">Зберегти зміни</button>
        </div>
    </div>
  </div>
</div>

<!-- Модальне вікно для підтвердження видалення проекту -->
<div class="modal fade" id="deleteProjectModal" tabindex="-1" aria-labelledby="deleteProjectModalLabel" aria-hidden="true">
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
```

```
                    <h5 class="modal-title" id="deleteProjectModalLabel">Видалення проекту</h5>
                    <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>
                </div>
                <div class="modal-body">
                    <p>Ви впевнені, що хочете видалити цей проект?</p>
                    <input type="hidden" id="deleteProjectId">
                </div>
                <div class="modal-footer">
                    <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Скасувати</button>
                    <button type="button" class="btn btn-danger" id="confirmDeleteProject">Видалити</button>
                </div>
            </div>
        </div>
    </div>

    <script>
        $(document).ready(function() {
            // Додавання проекту
            $('#saveProject').click(function() {
                var formData = $('#addProjectForm').serialize();
                $.ajax({
                    url: 'projects.php',
                    type: 'POST',
                    data: formData + '&action=add',
                    success: function(response) {
                        var data = JSON.parse(response);
                        if (data.success) {
```

```
                location.reload();
            }
        }
    });
});

// Редагування проекту
$('.edit-project').click(function() {
    var projectId = $(this).data('id');
    $.ajax({
        url: 'projects.php',
        type: 'POST',
        data: { id: projectId, action: 'edit' },
        success: function(response) {
            var project = JSON.parse(response);
            $('#editProjectId').val(project.id);
            $('#editName').val(project.name);
            $('#editDescription').val(project.description);
            $('#editStartDate').val(project.start_date);
            $('#editEndDate').val(project.end_date);
            $('#editStatus').val(project.status);
            $('#editProjectModal').modal('show');
        }
    });
});

// Оновлення проекту
$('#updateProject').click(function() {
    var formData = $('#editProjectForm').serialize();
    $.ajax({
        url: 'projects.php',
```

```
        type: 'POST',
        data: formData + '&action=update',
        success: function(response) {
            var data = JSON.parse(response);
            if (data.success) {
                location.reload();
            }
        }
    });
});

// Видалення проекту
$('.delete-project').click(function() {
    var projectId = $(this).data('id');
    $('#deleteProjectId').val(projectId);
    $('#deleteProjectModal').modal('show');
});

$('#confirmDeleteProject').click(function() {
    var projectId = $('#deleteProjectId').val();
    $.ajax({
        url: 'projects.php',
        type: 'POST',
        data: { id: projectId, action: 'delete' },
        success: function(response) {
            var data = JSON.parse(response);
            if (data.success) {
                location.reload();
            }
        }
    });
```

```
        });
    });
</script>

<?php include 'templates/footer.php'; ?>
```

UNICHECK
by Turnitin

Ім'я користувача:
Інформаційних систем в економіці Шкуратовська Те...

ID перевірки:
1016248112

Дата перевірки:
13.05.2024 20:10:32 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
14.05.2024 06:54:12 EEST

ID користувача:
100005745

Назва документа: Кукса К. ІУС-601

Кількість сторінок: 70 Кількість слів: 13567 Кількість символів: 114466 Розмір файлу: 1.42 MB ID файлу: 1016033378

4.7%
Схожість

Найбільша схожість: 0.52% з Інтернет-джерелом (https://kneu.edu.ua/userfiles/Economic_Department/KYRS%20PRAK%2

2.28% Джерела з Інтернету 219 ..... Сторінка 72

4.57% Джерела з Бібліотеки 570 ..... Сторінка 74

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВАДИМА ГЕТЬМАНА

Навчально-науковий інститут
«Інститут інформаційних технологій в економіці»

Кафедра інформаційних систем в економіці

ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА
«Інформаційні управляючі системи і технології»

галузь знань 12 Інформаційні технології

Спеціальність 122 Комп'ютерні науки

Форма навчання: заочна

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:
«Інформаційна система управління процесами тестування та виявлення дефектів програмного забезпечення»

Здобувача **Кукси Катерини Миколаївни**

Науковий керівник: д.е.н., професор Мозгаллі О. П.

_____
(підпис)

Робота допущена до захисту перед екзаменаційною комісією з атестації здобувачів вищої освіти (ЕК)

Завідувач кафедри: к.е.н., доцент Тішков Б. О.

_____
(підпис)

Київ 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Київський національний економічний університет імені Вадима Гетьмана
Кафедра бізнес-економіки та підприємництва
Інститут інноваційного підприємництва КНЕУ імені Вадима Гетьмана
Український національний офіс інтелектуальної власності та інновацій
Max Planck Institute for Innovation and Competition (Німеччина)
Uniwersytet Jagielloński (Польща)
Szkoła Główna Handlowa w Warszawie (Польща)
НДІ Інтелектуальної власності Національної академії правових наук України
Інститут експериментальної патології, онкології і радіобіології
ім. Р.Є. Кавецького НАН України
НТУУ «Київський політехнічний інститут імені І. Сікорського»
AgileLeanHouse (Данія)

IX МІЖНАРОДНА НАУКОВО-ПРАКТИЧНА ІНТЕРНЕТ-КОНФЕРЕНЦІЯ «ІННОВАЦІЙНЕ ПІДПРИЄМНИЦТВО: СТАН ТА ПЕРСПЕКТИВИ РОЗВИТКУ»

29 березня 2024 року
Київ, КНЕУ

УДК

Рекомендовано до друку Науково-експертною Радою
Київського національного економічного університету імені Вадима Гетьмана»
Протокол №

Програмний комітет:

Голова програмного комітету: Рєпіна І.М., д.е.н., професор, завідувач бізнес-економіки та підприємництва ДВНЗ КНЕУ імені Вадима Гетьмана; Члени програмного комітету: Лаврененко В.В., к.е.н., доцент, професор кафедри бізнес-економіки та підприємництва ДВНЗ КНЕУ імені Вадима Гетьмана; Петренко Л.А., д.е.н., професор, професор кафедри бізнес-економіки та підприємництва ДВНЗ КНЕУ імені Вадима Гетьмана.; Daria Kim - Senior Research Fellow (MA, LLM, Dr iur) Max Planck Institut für Innovation und Wettbewerb; Т.В. П'ятчаніна, к.б.н, с.н.с., завідуюча відділу менеджменту наукових досліджень та інновацій, ІЕПОР ім. Р.Є. Кавецького НАН України; Бутнік-Сіверський О. Б., д.е.н., професор, завідувач кафедри економіки, обліку та фінансів Інституту післядипломної освіти Національного університету харчових технологій; О.Ф. Дорошенко, к.ю.н., директор НДІ інтелектуальної власності НАПрН України; О.Ю. Кашинцева, к.ю.н., доцент, завідувачка відділом дослідження інтелектуальної власності та прав людини в сфері охорони здоровя НДІ інтелектуальної власності НАПрН України;

Інноваційне підприємництво: стан та перспективи розвитку [Електронний ресурс]:
І Зб. матеріалів IX Міжнар. наук.-практ. конференції. — К.: КНЕУ, 2024. — 478с.
ISBN

Досліджено потенціал інноваційного підприємництва у відновленні економіки України, а також діджітал-технології в підприємництві в епоху економіки даних; висвітлено розвиток академічного підприємництва, бізнес-акселерацію та жіноче підприємництво; виявлено перспективи розвитку інноваційного підприємництву у сфері охорони здоров'я і забезпечення доступності ліків; висвітлено менеджмент інтелектуальної власності: сучасні виклики та можливості; наведено досвід розвитку підприємницької освіти та професійного розвиток підприємців.

УДК 330.341.1:338.22(082)
ББК 65.291.551я43

ISBN

ЗМІСТ

JEL Classification: L26, M13, O31, O33

Кукса Катерина
магістрант кафедри економіки та підприємництва,
ДВНЗ «КНЕУ ім. Вадима Гетьмана»,
Kateryna Kuksa
PhD student at the Department of Economics and Entrepreneurship,
Vadym Hetman Kyiv National Economic University,
kateryna.kuksa@kneu.ua

ДИДЖИТАЛ-ПІДПРИЄМНИЦТВО, ЦИФРОВІ БІЗНЕС-МОДЕЛІ ТА ШТУЧНИЙ ІНТЕЛЕКТ: НОВІ ГОРИЗОНТИ РОЗВИТКУ

DIGITAL ENTREPRENEURSHIP, DIGITAL BUSINESS MODELS AND ARTIFICIAL INTELLIGENCE: NEW HORIZONS OF DEVELOPMENT

Анотація. У статті досліджено сучасні тенденції розвитку диджитал-підприємництва, цифрових бізнес-моделей та штучного інтелекту. Проаналізовано вплив цифрових технологій на трансформацію підприємницької діяльності та формування нових бізнес-моделей. Визначено ключові напрямки застосування штучного інтелекту в підприємництві та обґрунтовано його роль у підвищенні ефективності та конкурентоспроможності бізнесу.
Abstract. The article explores the current trends in the development of digital entrepreneurship, digital business models and artificial intelligence. The impact of digital technologies on the transformation of entrepreneurial activity and the formation of new business models is analyzed. The key areas of artificial intelligence application in entrepreneurship are identified and its role in improving business efficiency and competitiveness is substantiated.

Стрімкий розвиток цифрових технологій, зокрема штучного інтелекту (ШІ), відкриває нові можливості для підприємницької діяльності та радикально трансформує традиційні бізнес-моделі. Диджитал-підприємництво, що базується на використанні цифрових платформ, великих даних, хмарних обчислень та ШІ, стає потужним драйвером інновацій та економічного зростання.

Цифрові бізнес-моделі дозволяють підприємцям суттєво підвищити ефективність операційних процесів, персоналізувати взаємодію з клієнтами, автоматизувати рутинні завдання та приймати більш обґрунтовані управлінські рішення на основі аналізу даних. Такі моделі, як електронна комерція, платформна економіка, підписка та freemium, дають змогу масштабувати бізнес, виходити на глобальні ринки та створювати нові джерела доходів.

Штучний інтелект відіграє ключову роль у трансформації підприємницької діяльності, надаючи потужні інструменти для аналізу великих даних, прогнозування

попиту, персоналізації маркетингу, оптимізації ланцюгів поставок та автоматизації клієнтського сервісу. Застосування ШІ дозволяє підприємцям приймати більш ефективні рішення, знижувати витрати та створювати інноваційні продукти і послуги.

Серед ключових напрямків застосування ШІ в диджитал-підприємництві можна виділити:

- персоналізація клієнтського досвіду. ШІ-алгоритми дозволяють аналізувати поведінку користувачів, їх вподобання та історію покупок, щоб пропонувати персоналізовані рекомендації, релевантний контент та targeted advertising. Чат-боти та віртуальні асистенти на основі ШІ забезпечують цілодобову підтримку клієнтів та миттєве реагування на запити.
- предиктивна аналітика та оптимізація бізнес-процесів. ШІ-моделі дають змогу прогнозувати попит, планувати виробництво, оптимізувати ланцюги поставок та управляти запасами. Предиктивне обслуговування обладнання на основі ШІ дозволяє зменшити простої та підвищити ефективність виробництва.
- автоматизація рутинних завдань. ШІ-системи можуть автоматизувати процеси обробки документів, розпізнавання зображень, відповідей на запити клієнтів, що дозволяє вивільнити час працівників для більш творчих та стратегічних завдань. Роботизована автоматизація процесів (RPA) на основі ШІ підвищує швидкість та точність виконання повторюваних операцій.
- безпека та запобігання шахрайству. ШІ-алгоритми допомагають виявляти потенційні загрози безпеці, такі як шахрайські транзакції, вторгнення в мережу або витоки даних. Біометрична автентифікація на основі ШІ підвищує надійність та зручність доступу до цифрових сервісів.
- генерація контенту та дизайн продуктів. ШІ-моделі можуть створювати унікальний текстовий, графічний та відеоконтент, адаптований до потреб конкретної аудиторії. Генеративний дизайн на основі ШІ дозволяє швидко створювати численні варіанти дизайну продуктів з оптимальними характеристиками.

Разом з тим, впровадження ШІ в підприємницьку діяльність також пов'язане з певними викликами та ризиками. Це, зокрема, питання забезпечення безпеки та конфіденційності даних, необхідність розвитку цифрових компетенцій персоналу, потенційні етичні проблеми використання ШІ. Підприємцям необхідно враховувати ці аспекти при розробці відповідальних та стійких цифрових бізнес-моделей.

У довгостроковій перспективі, синергія між диджитал-підприємництвом, цифровими бізнес-моделями та штучним інтелектом матиме трансформаційний вплив на структуру економіки та суспільства. Це вимагатиме адаптації регуляторної бази, розвитку цифрових екосистем та партнерств, а також відповідального впровадження ШІ з урахуванням суспільних інтересів та цінностей.

Для України розвиток диджитал-підприємництва на основі ШІ є особливо актуальним в контексті післявоєнного відновлення та модернізації економіки. Це дасть змогу підвищити продуктивність та конкурентоспроможність вітчизняного бізнесу, залучити іноземні інвестиції та інтегруватися до глобальних ланцюгів створення вартості.

Важливими кроками у цьому напрямку є розбудова цифрової інфраструктури, розвиток цифрових навичок та компетенцій, стимулювання НДДКР та інновацій у сфері ШІ, а також формування сприятливого бізнес-середовища для диджитал-підприємництва.

Література

1. Andrusiak, N. O., Kraus, N. M., & Kraus, K. M. (2020). Digital Cubic Space as a New Economic Augmented Reality. *Sci. innov., 16(3)*, 92-105. DOI: 10.15407/scine16.03.092
2. Болдирєва, Л. М., Краус, Н. М., & Краус, К. М. (2019). Цифрові компетенції в сфері вищої освіти: задум, реалізація, результат. *Держава та регіон. Серія: Економіка та підприємництво, 1 (106)*, 4–9.
3. Brytskyi, R. M., Kraus, N. M., & Kraus, K. M. (2021). Entrepreneurship Development in the Sharing Economy. In *The World Economy and International Economic Relations*(pp. 45-55). Publishing House "Baltija Publishing". DOI: 10.30525/978-9934-26-079-7-27
4. Краус, Н. М., Краус, К. М., & Манжура, О. В. (2021). Екосистема гіг-економіки та підприємницького університету: еволюційна синергетика «вірусу інновацій» та «цифрового стрибка». *Ефективна економіка, 2*. DOI: 10.32702/2307-2105-2021.2.3
5. Marchenko, O., Kraus, N., & Kraus, K. (2021). The Impact of Servation on the Results of Economic Digital Entrepreneurship Activities. *Ukraine in the Context of Global and National Modern Servisation Processes and Digital Economy: Monograph*, Praha: OKTAN PRINT, 81-91. DOI: 10.46489/UITCOG0909
6. Odnorog, M., Kraus, N., & Kraus, K. (2019). The features of entrepreneurial interactions in the interactions in the agricultural sector in terms of institutional transformation. *Baltic Journal of Economic Studies, 4*, 171-181. DOI: 10.30525/2256-0742/2019-5-4-171-181

References

1. Andrusiak, N. O., Kraus, N. M., & Kraus, K. M. (2020). Digital Cubic Space as a New Economic Augmented Reality. *Sci. innov., 16(3)*, 92-105. DOI: 10.15407/scine16.03.092
2. Boldyreva, L. M., Kraus, N. M., & Kraus, K. M. (2019). Digital competencies in the field of higher education: design, implementation, result. *State and region. Series: Economics and Entrepreneurship, 1 (106)*, 4–9.
3. Brytskyi, R. M., Kraus, N. M., & Kraus, K. M. (2021). Entrepreneurship Development in the Sharing Economy. In *The World Economy and International Economic Relations*(pp. 45-55). Publishing House "Baltija Publishing". DOI: 10.30525/978-9934-26-079-7-27
4. Kraus, N. M., Kraus, K. M., & Manzhura, O. V. (2021). Ecosystem of gig-economy and entrepreneurial university: evolutionary synergy of "virus innovation" and "digital jump". *Efektyvna ekonomika, 2*. DOI: 10.32702/2307-2105-2021.2.3
5. Marchenko, O., Kraus, N., & Kraus, K. (2021). The Impact of Servation on the Results of Economic Digital Entrepreneurship Activities. *Ukraine in the Context of Global and National Modern Servisation Processes and Digital Economy: Monograph*, Praha: OKTAN PRINT, 81-91. DOI: 10.46489/UITCOG0909

6. Odnorog, M., Kraus, N., & Kraus, K. (2019). The features of entrepreneurial interactions in the interactions in the agricultural sector in terms of institutional transformation. *Baltic Journal of Economic Studies, 4*, 171-181. DOI: 10.30525/2256-0742/2019-5-4-171-181