Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Дослідження методів комплексного тестування корпоративної інформаційної системи

(тема)

Виконав:

студент II курсу, групи СПм-21-1

Мельникова К.С.
(прізвище, ініціали)

Спеціальність

123 «Комп'ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма

Системне програмування
(повна назва освітньої програми)

Керівник: проф. Рубан І.В.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ _____ Коваленко А.А.
(підпис) (прізвище, ініціали)

2022 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Мельниковій Катерині Сергіївні
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження методів комплексного тестування корпоративної інформаційної системи

затверджена наказом по університету від " 07 " листопада 2022 р. № 1454Ст

2. Термін подання студентом роботи до екзаменаційної комісії 13 грудня 2022 р.

3. Вхідні дані до роботи

4. Перелік питань, що потрібно опрацювати у роботі

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) _____

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання	07.11	
2	Аналіз і пошук інформації щодо поставленої	07.11 – 11.11	
	задачі		
3	Розробка алгоритму виконання завдання	11.11 – 17.11	
4	Тестування алгоритму	17.11 – 19.11	
5	Виконання завдання	19.11 – 05.12	
6	Підготовка звіту	06.12 – 10.12	
7	Отримання рецензії	11.12	
8	Проходження нормоконтролю	13.12	

Дата видачі завдання 07 листопада 2022 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Рубан І.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 119с., 58 с., 7 табл., 2 дод., 55 джерел.

КОМП’ЮТЕРНА СИСТЕМА, ТЕСТУВАННЯ, СЕРВЕР, НАДІЙНІСТЬ, ПРОДУКТИВНІСТЬ, БАЗА ДАНИХ, АНАЛІЗ, ЖИТТЭВИЙ ЦИКЛ, ВЕБ, БРАУЗЕР.

Метою кваліфікаційної роботи є аналіз методів тестування та розробка сценарію комплексного тестування для підвищення продуктивності та безпеки корпоративних інформаційних систем.

Позначена мета визначила об'єкт та предмет дослідження.

Об'єкт дослідження – методи комплексного тестування корпоративних інформаційних систем.

Предмет дослідження – розробка сценарію комплексного тестування для підвищення продуктивності та надійності корпоративних інформаційних систем.

Гіпотезою дослідження є припущення щодо можливості створення на основі існуючих методів тестування сценарію комплексного тестування корпоративних інформаційних систем для підвищення їхньої продуктивності та інформаційної безпеки.

У ході виконання кваліфікаційної роботи розроблено теоретичні положення концепції аналізу та порівняння методів тестування інформаційних систем, що дозволяють здійснювати вибір найбільш відповідного методу для практичного використання, покладеного в основу розробки сценарію комплексного тестування, що дозволить суттєво скоротити витрати на тестування ІС та підвищити якість її функціонування.

ABSTRACT

Master’s thesis: 119 pages, 58 figures, 7 tables, 2 appendices, 55 sources.

COMPUTER SYSTEM, TESTING, SERVER, ADDITION, PRODUCTIVITY, DATABASE, ANALYSIS, LIFE CYCLE, WEB, BROWSER.

The method of qualification work is the analysis of testing methods and the development of a comprehensive testing scenario for improving the productivity and security of corporate information systems.

The meta was assigned to the object and the subject of the follow-up.

The object of follow-up is a method of complex testing of corporate information systems.

The subject of the study is the development of a comprehensive testing scenario for improving the productivity and reliability of corporate information systems.

The hypothesis is based on the possibility of creating on the basis of basic methods in the scenario of integrated testing of corporate information systems to improve their productivity and information security.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПП – програмний продукт;

ПЗ – програмне забезпечення;

КІС – корпоративні інформаційні системи;

ТЗ – технічне завдання;

ІС – інформаційна система;

БД – база даних;

СУБД – система управління базами даних;

STLC – Software Testing Life Cycle - життєвий цикл тестування програмного забезпечення;

QA – quality assurance (контроль якості);

XP – Extreme programming (екстремальне програмування);

OWA – Outlook Web App.

ВСТУП

Інтенсивний розвиток і все більше впровадження досягнень інформаційних технологій (ІТ) вимагає від проектувальників та розробників інформаційних систем (ІС) створення ефективних та якісних програмних додатків. Це говорить про те, що перевірка якості є важливим процесом у створенні ІС, при цьому питання та проблеми тестування висвітлені у доступній літературі менше, ніж будь-який інший аспект розробки програмного забезпечення.

Аналіз публікацій показав, що сенс терміну «тестування» та визначення його передається різними авторами як: «тестування є процесом, що демонструє відсутність помилок у програмі», «мета тестування – показати, що програма коректно виконує передбачені функції» , «Тестування - це процес, що дозволяє переконатися в тому, що програма виконує своє призначення» і є недостатньо коректними з точки зору економічної складової (тобто ціни розробки програмного продукту (ПП)). Найчастіше програміст не в змозі оцінити результати своєї роботи з погляду ефективності ІВ, де вона використовується, тобто. Він практично нічого не може сказати про те, наскільки протестовано код його програми. Крім того, дуже часто розробникам, менеджерам проектів, керівнику фірми необхідно вирішувати питання, пов'язані зі скороченням витрат на виробництво ПП та підвищення якості програмного забезпечення (ПЗ). Основним способом вирішення цих проблем є тестування ПЗ ІС. Процес тестування включає вирішення питань не тільки технічного характеру (організація ефективного процесу тестування, визначення часу тестування, використання або невикористання інструментальних засобів тощо), але і питань економічного і психологічного характеру.

Аналіз досліджень та публікацій у галузі тестування показав, що розгляд таких аспектів тестування [1], як технічні, економічні, психологічні, представляє значний практичний інтерес.

Відповідно до вищесказаного, метою є дослідження тестів для перевірки програмних додатків ІС з точки зору психологічного, економічного та технічного підходів. Великий внесок у вирішення різних аспектів проблеми тестування та програмної надійності зробили вітчизняні вчені: Єршов А.П., Глушков В.М., Лебедєв С.О., Михалевич В.С., Сергієнко І.В., Ющенко К.Л., Івашко А.І., Воєводін Ю.М., Болотських М.С., Стадниченко Ю.І., Татаренко Л.С., а також зарубіжні фахівці: Andrews A., Offut J., Changyou Xing, Heiskanen H., Maunumaa M., Honlin Zh., Wenbo X., Makinen M. та інші.

Метою кваліфікаційної роботи є аналіз методів тестування та розробка сценарію комплексного тестування для підвищення продуктивності та безпеки корпоративних інформаційних систем.

Позначена мета визначила об'єкт та предмет дослідження.

Об'єкт дослідження – методи комплексного тестування корпоративних інформаційних систем.

Предмет дослідження – розробка сценарію комплексного тестування для підвищення продуктивності та надійності корпоративних інформаційних систем.

Гіпотезою дослідження є припущення щодо можливості створення на основі існуючих методів тестування сценарію комплексного тестування корпоративних інформаційних систем для підвищення їхньої продуктивності та інформаційної безпеки.

Ґрунтуючись на цій гіпотезі, застосування методів будуть найбільш ефективними, якщо:

- визначено поняття комплексного тестування ІС;
- вибрані методи тестування, що входять до складу комплексного;
- змодельовано та реалізовано сценарій проведення комплексного тестування, що дозволяє максимально перевірити ІС на помилки у її продуктивності та управлінні інформаційної безпеки.

Для того, щоб досягти мети, необхідно вирішити завдання:

1 розглянути теоретичні засади тестування інформаційних систем, що

дозволяють здійснювати аналіз наявних методів тестування інформаційних систем;

2 проаналізувати на основі розробленої теорії практичні методи тестування програмного забезпечення ІС;

3 розробити сценарій комплексного тестування, що дозволяє з одного боку оцінити продуктивність, з другого – її інформаційну безпеку;

4 підтвердити шляхом експерименту ефективність реалізованого сценарію та результати його запровадження, використовуючи процес апробації.

Наукова новизна дослідження полягає в тому, що в ньому визначено основні критерії ефективності тестування, які застосовуються для оцінки продуктивності та надійності ІС; розроблено сценарій комплексного тестування ІС, що дозволяє на основі визначення типів помилок, що найчастіше зустрічаються, задавати якісні тести, що забезпечують мінімізацію всіляких втрат у бізнес-середовищі організації.

Практична значущість дослідження полягає у розробці теоретичних положень концепції аналізу та порівняння методів тестування інформаційних систем, що дозволяють здійснювати вибір найбільш відповідного методу для практичного використання, покладеного в основу розробки сценарію комплексного тестування, що дозволить суттєво скоротити витрати на тестування ІС та підвищити якість її функціонування.

Основні етапи дослідження в кілька етапів:

На першому етапі (констатуючому етапі) - здійснювався аналіз інформації з висунутої проблеми, на підставі якого була сформульована тема дослідження темі дослідження, ставилася гіпотеза дослідження, визначалися мета та завдання дослідження.

Другий етап (пошуковий етап) – здійснювалось моделювання сценарію, що базується на методах комплексного тестування корпоративних інформаційних систем (КІС).

Третій етап (експериментальний) – здійснювалася експлуатація

розробленого сценарію на декількох прикладах корпоративних систем, що дало впевненість у правильних висновках дослідження, проводилася оцінка результатів та були сформульовані висновки про отриманий результат проведеного дослідження.

На захист виноситься:

- методи комплексного тестування корпоративної інформаційної системи;
- розроблений сценарій проведення комплексного тестування КІС;
- результати експлуатації, що демонструють ефективність розробленого сценарію комплексного тестування.

У першому розділі розглядаються теоретичні засади тестування корпоративних систем. У другому розділі буде представлено аналіз методів тестування ІС. У третій главі буде наведено процес обробки реалізованого сценарію проведення комплексного тестування з демонстрацією отриманих результатів. Четвертий розділ буде присвячено процесу оцінки ефективності реалізованого сценарію комплексного тестування ІС.

У висновку підбиватимуться підсумки виконаної роботи.

1 ТЕОРЕТИЧНІ ОСНОВИ ТЕСТУВАННЯ КОРПОРАТИВНИХ ІНФОРМАЦІЙНИХ СИСТЕМ

1.1 Життєвий цикл тестування корпоративної ІС

КІС з погляду системного аналізу відносяться до складних систем, оскільки вони побудовані з безлічі різних підсистем: серверів баз даних (БД), серверів додатків, серверів, що кешують, балансувальників, серверів резервного копіювання, безпеки тощо. У добре спроектованій системі великий ступінь взаємної інтеграції, яка збільшує ризик виходу з ладу однієї або кількох підсистем КІС, за яким слідує з ладу інших раніше працездатних підсистем і так далі аж до повного руйнування всієї системи. Без спеціальних засобів обробки нештатних ситуацій КІС виявиться нестійкою до помилок і не зможе гарантувати закладені у технічні вимоги та угоду про рівень послуг вимоги щодо часу відновлення до працездатного стану у разі виникнення передбачуваних та непередбачених помилок.

У зв'язку з цим розробка методик (сценаріїв) тестування КІС [2], проведеного на усіх етапах її життєвого циклу, є актуальною.

Життєвий цикл тестування програмного забезпечення (STLC) – це процес тестування, який виконується систематично та в плановому порядку. У процесі STLC для покращення якості продукту виконуються різні дії. У життєвому циклі тестування програмного забезпечення (STLC) передбачені такі етапи зі своїми критеріями входу та результатами (таблиці 1.1 – Фази циклу STLC та результати заходів):

- аналіз вимог – на цьому етапі команда забезпечення якості (QA) розуміє вимоги з точки зору того, що ми тестуватимемо, і з'ясуємо вимоги, що тестуються. Вимоги можуть бути функціональними або нефункціональними, такими, як продуктивність, тестування безпеки;

- планування випробувань – на цьому етапі визначаються зусилля та

оцінки витрат для проекту:

- розробка тестового прикладу – на цьому етапі STLC команда тестування записує докладні тестові приклади. Поряд із тестовими наборами команда тестування також готує тестові дані, якщо такі необхідні для тестування;

- налаштування тестового середовища, яке визначає умови тестування програмного забезпечення;

- виконання тесту – на цьому етапі команда тестування розпочинає виконання тестових прикладів на основі підготовленого планування тестів та підготовлених тестових прикладів на попередньому етапі. Якщо якийсь із тестових випадків заблокований через якийсь дефект, то такі тестові випадки можуть бути позначені як заблоковані, тому ми можемо отримати звіт на основі того, скільки тестових прикладів пройшло, провалилося, заблоковано або не виконано тощо [3]. Після усунення дефектів ті самі тести з помилками або блокування можуть бути виконані знову для повторного тестування функціональності;

- закриття циклу випробувань – на цьому етапі аналізуються помилки, розподіляються дефекти на кшталт і складності.

Таблиця 1.1 – Фази цику STLC та результати заходів

	Критерії входу	Заходи	Результат
Аналіз вимог	- технічні вимоги; - архітектурний додаток; - чітко визнані критерії прийому.	- зустріч з BA, System Architecture, Technical Manager/Lead тощо; - список виконаних тестів; - визначити пріоритети тестування; - обране середовище тестування.	Тестовані вимоги

Продовження таблиці 1.1

Планування випробувань	- документи з вимогами; - автоматизація техніко-економічного обґрунтування.	- визначити мету та обсяг проекту; - перелічити типи тестування; - оцінка зусиль з тестування та планування ресурсів; - вибір інструменту та середовище тестування; - розклад випробувань; - визначити контрольні процедури; - визначити ролі та обов'язки; - перерахувати результати тестування; - визначити критерії входу, відновлення та виходу; - визначити ризик.	План тестування/документ стратегії тестування.
Розробка тестового прикладу	- документи з вимогами (оновлена версія). - автоматизація техніко-економічного обґрунтування.	- підготовка тестових випадків; - підготовка сценаріїв автоматизації тестування (за потреби); - підготовка необхідних тестових даних для виконання тестових випадків.	- тестові випадки; - тестові дані; - тестування сценаріїв автоматизації (якщо потрібно).

Продовження таблиці 1.1

Налаштування тестового середовища [4]	Доступні: план випробувань, smoke - тест та набір тестових даних.	- проаналізуйте вимоги та підготуйте список програмного та апаратного забезпечення, необхідного для налаштування тестового середовища; - налаштуйте тестове середовище; - після налаштування середовища тестування виконайте контрольні тести, щоб перевірити готовність тестового середовища.	- готове тестове середовище та тестові дані; - результат smoke - тесту
Виконання тесту	- план тестування чи документ про стратегію тестування; - тестові випадки; - тестові дані.	- на основі планування тестування виконайте контрольні приклади; - відзначити стан тестових випадків, таких як Pass, Fail, Blocked тощо; - призначити ідентифікатор помилки всім невдалих та заблокованих тестових випадків; - Здійснити повторне тестування; - відстежувати дефекти до закриття.	Звіти про виконання тесту та дефекти

Продовження таблиці 1.1

Закриття циклу випробувань	- виконання тестового прикладу; - звіт про виконання тесту; - звіт про дефекти.	- оцінити критерії завершення циклу на основі охоплення тестування, якості, вартості, часу, критичних бізнес-цілей та ПЗ; - підготувати показники тесту на основі вищезгаданих параметрів; - підготувати звіт про закриття тесту; - поділитись передовим досвідом для будь-яких подібних проектів у майбутньому.	Звіт про закриття тесту та метрики тестів

Сценарії тестування STLC повинні сприяти пошуку непередбачених помилок/проблем у роботі КІС на всіх етапах життєвого циклу, що виражається замкненою послідовністю дій (рисунок 1.1):

- стадія 1 – загальне планування та аналіз вимог [5, 6] – для визначення методів та видів тестування, обмежень та ризиків, належить тестувати; наявності необхідного інструментарію тощо;
- стадія 2 – уточнення критеріїв приймання – для визначення/уточнення метрик та ознак можливості/необхідності початку, зупинення та відновлення тестування, завершення або припинення тестування;
- стадія 3 – уточнення стратегії тестування - для розгляду та уточнення актуальних для поточної ітерації частин стратегії тестування;
- стадія 4 – розробка тест-кейсів [7] – для розробки, перегляду,

уточнення, доопрацювання, переробки з тест-кейсами/тестовими сценаріями [45];

- стадія 5 – виконання тест-кейсів – для безпосереднього виконання сценарію тестування;

- стадія 6 – фіксація знайдених дефектів - на формування розуміння проблеми та уточнення важливості та терміновості проведення тестування;

- стадія 7 – аналіз результатів тестування – для обробки отриманих результатів, щоб приймати рішення щодо ще однієї ітерації;

- стадія 8 – звітність - для фіксування проміжних та кінцевих варіантів тестування.

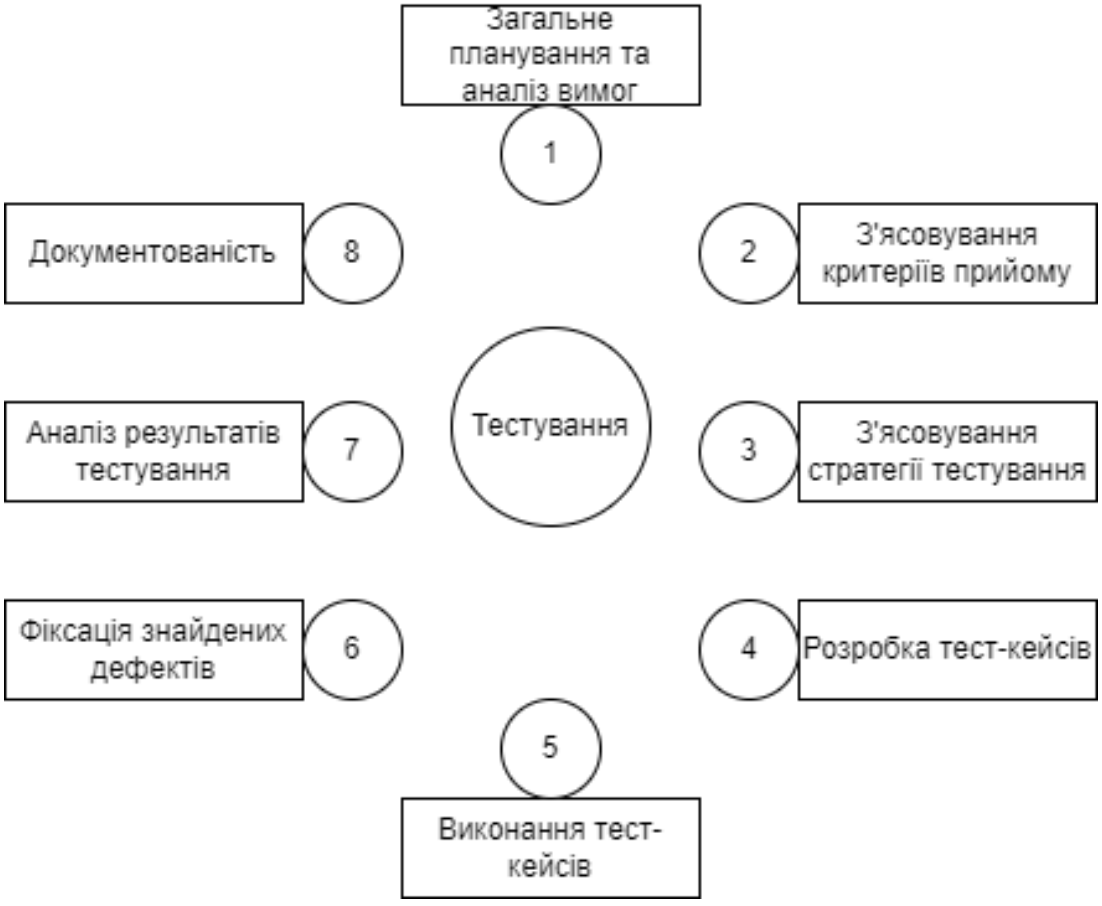

Рисунок 1.1 – Життєвий цикл тестування

Кожна з виділених стадій тестування може розглядатися як така, що забезпечує створення певного проміжного продукту – функціональної групи

програм або програмного засобу з деякими обмеженими характеристиками якості. Ці характеристики виділяються та деталізуються на основі первинного технічного завдання та специфікації вимог на ПС. У процесі проектування ПС вони уточнюються та конкретизуються у специфікаціях вимог на групи програм та їх компоненти. В результаті створюється сукупність еталонів, що мають послідовно розширюються номенклатуру і набори значень показників якості, яким повинні відповідати компоненти, що налагоджуються і випробовуються, на кожній стадії тестування.

Таким чином, пройшовши основні стадії життєвого циклу тестування ІС, можна з упевненістю стверджувати про працездатність програмного продукту.

1.2 Принципи та основні етапи комплексного тестування КІС

Тестування КІС охоплює основні стадії життєвого циклу, аналогічний до послідовності процесів розробки ПЗ [8]: постановка задачі для тесту, проектування, написання тестів, тестування тестів, виконання тестів та вивчення результатів тестування. Щоб орієнтуватися у тестах, варто розглянути два основні підходи.

Перший підхід орієнтований вивчення логіки програмного забезпечення КІС під час проектування тестів. У процесі проектування тестів необхідно передбачити, щоб кожна команда умовного переходу виконувалася кожному напрямі хоча б раз, тобто. Необхідно перевірити кожну галузь алгоритму, визначальну певний шлях.

Другий підхід тестування ґрунтується на виконанні фундаментального принципу функціонування КІС: віддача має перевищувати витрати. Ця віддача вимірюється ймовірністю того, що тест виявить невиявлену раніше помилку. Витрати вимірюються часом та вартістю підготовки, виконання та перевірки результатів тесту. Кожен тест повинен бути представлений деяким класом вхідних значень таким чином, щоб його правильне виконання створювало переконаність у тому, що програма виконуватиметься правильно для певного

класу вхідних даних.

Якщо врахувати, що КІС – це не тільки ті, що використовуються в її складі програмні компоненти, а й апаратне та організаційне забезпечення, то й у результатах її випробувань мають бути відображені показники обраних серверів, робочих станцій, мережевого обладнання (їх надійність та продуктивність), а також ефективність розробленого регламенту експлуатації системи. У зв'язку з цим виникає необхідність у проведенні комплексного тестування на відповідність усім вимогам, що висуваються.

Комплексне тестування КІС складається з наступних етапів [9]:

1 детальне вивчення проекту системи та її експлуатаційних документів;

2 створення та впровадження автоматизованої системи відстеження помилок (bug tracking);

3 безпосереднє тестування роботи ІС, яке включає:

- тестування роботи всіх модулів ІС;
- перевірка внутрішньосистемних зв'язків;
- тестування устаткування, зокрема наявність необхідних ліцензій;
- перевірка програмного забезпечення, у тому числі перевірка коду;
- тестування продуктивності та максимальних навантажень ІС;
- тестування на відмови: несподівані програмні збої, вихід з ладу модулів системи, людський фактор та ін;
- тестування на захищеність ІС - включає комплекс досліджень з виявлення способів злому системи і витоків інформації;
- загальна перевірка роботи системи.

4 тестування роботи системи керування it-структурою;

5 тестування персоналу;

6 аналіз отриманих даних та вироблення стратегії з виправлення виявлених помилок [10].

Комплексне тестування призначене для тестування всіх функцій повністю зібраної системи. Воно спрямовано пошук невідповідності системи її вихідним цілям. Таким чином, у комплексному тестуванні бере участь КІС, її цілей та вся

документація, яка поставлятиметься разом із системою.

1.3 Методологія тестування КІС

Методологія тестування є стратегією та підходом для тестування ІС, щоб гарантувати, що програмний продукт придатний для експлуатації [11].

Методики тестування включають тестування того, що ІС працює відповідно до специфікації, не має небажаних побічних ефектів при використанні способами, що виходять за межі проектних параметрів, і в гіршому випадку буде стійким до відмови.

Методології тестування ІС [12] – це різні підходи та способи забезпечення того, щоб ІС була повністю протестована. Методології тестування охоплюють усі: від модульного тестування окремих модулів, інтеграційного тестування всієї ІС до спеціалізованих форм тестування, таких як безпека та продуктивність.

Оскільки ІС стають все більш складними та взаємопов'язаними, а також з великою кількістю різних платформ та пристроїв, які необхідно протестувати, важливо мати надійну методологію тестування, щоб переконатися, що програмні продукти/ІС були повністю протестовані, що вони відповідають зазначеним вимогам і можуть успішно працювати у всіх очікуваних середовищах з необхідною зручністю використання та безпекою.

На рисунку 1.2 представлено загальну схему основних видів тестування ІС, передбачених методологіями тестування.

Рисунок 1.2 – Основні види тестування

Відповідно до даної схеми методологія [13, 14] тестування включає функціональне тестування, яке виконується з використанням функціональних специфікацій, наданих клієнтом, або з використанням специфікацій проектування, таких як сценарії використання, що надаються командою розробників. Функціональне тестування в методології тестування розбите на чотири компоненти: модульне тестування, інтеграційне тестування, системне тестування та приймальне тестування. Окремо про кожен з наведених типів:

- модульне тестування - тестування окремих програмних модулів чи компонентів, що входять до складу програми чи системи. Модульні тести зазвичай пишуться розробниками модуля, і в методології розробки, заснованої на тестуванні (такі як Agile, Scrum або XP), вони фактично пишуться до того, як модуль буде створений як частина специфікації. Кожна функція модуля тестується спеціальним модульним тестовим приладом, написаним тією ж мовою програмування, що сам модуль (рисунок 1.3).

```
public class SampleTestFixture
{
        [SetUp]
        public void Init()
        {
                //Do Nothing
        }

        /// <summary>
        /// Sample test that asserts a failure
        /// </summary>
        [
        Test,
        SpiraTestCase (<test case id>)
        ]
        public void _01_SampleFailure()
```

Рисунок 1.3 – Приклад модульного теста 1

- інтеграційне тестування [15] – тестування різних модулів/компонентів, які були успішно протестовані при інтегруванні разом для виконання конкретних завдань та заходів (також відомі як тестування сценарію). Це тестування зазвичай виконується за допомогою комбінації автоматичних функціональних тестів та ручного тестування;

- системне тестування включає тестування системи на наявність помилок. Даний вид тесту проводиться шляхом взаємодії з апаратними та програмними компонентами всієї системи, а потім здійснюється перевірка загалом (рисунок 1.4). Для цього типу тестування застосовується метод «чорної скриньки», де програмне забезпечення перевіряється на наявність очікуваних робочих умов, і навіть можливих виняткових і граничних умов.

Name	Execution Status	Planned Date	Release	Last Executed	Owner	Status	ID	Edit
-- Any --	-- Any --		-- Any --		-- Any --	-- Any --	TX	Edit
Exploratory Testing 2					Fred Bloggs	Deferred	TX:6	Edit
Testing Cycle for Release 1.0 7		4-Feb-2007	1.0.0.0	1-Dec-2003	Joe P Smith	In Progress	TX:1	Edit
Testing Cycle for Release 1.1 9		6-Feb-2007	1.1.0.0	1-Dec-2003	Joe P Smith	Not Started	TX:2	Edit
Testing New Functionality 4		9-Feb-2007	1.2.0.0		Fred Bloggs	In Progress	TX:5	Edit

Show 15 rows per page Displaying page 1 of 1

Рисунок 1.4 – Робота системного тестування

- приймальне тестування (рисунок 1.5) є заключним етапом функціонального тестування програмного забезпечення і включає перевірку того, що всі вимоги до продукту виконані, і що кінцеві користувачі та клієнти протестували систему [14], щоб переконатися, що вона працює належним чином і відповідає усім пред'явленим вимогам.

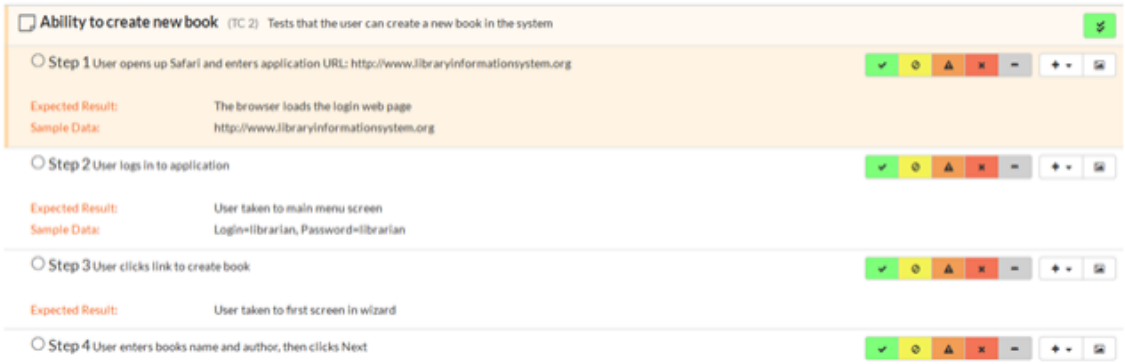

Рисунок 1.5 – Тестування прийому системи

Нефункціональне тестування включає тестування програми на відповідність нефункціональним вимогам, які зазвичай включають вимірювання/тестування програми на відповідність певним технічним якостям («здібностям»), наприклад: вразливість, масштабованість, зручність використання.

У більшості методологій тестування існує кілька різних типів тестування продуктивності (рисунок 1.6), наприклад:

- тестування продуктивності - це вимір поведінки системи при зростаючому навантаженні (як числа користувачів, так і обсягів даних);

- тестування навантаження - перевірка того, що система може працювати в необхідному режимі. Час відгуку, коли він піддається очікуваному навантаженню;

- стрес-тестування – це визначення точки відмови в системі, коли протестоване навантаження перевищує ту, яку вона може підтримувати;

- тестування сумісності (рисунок 1.7) перевіряє сумісність продукту або

програми з усіма вказаними операційними системами, апаратними платформами, веб-браузерами, мобільними пристроями та іншими розробленими сторонніми програмами (наприклад, плагінами браузера). Тести сумісності потрібні, щоб переконатися, що програмний продукт працює, як очікується, у всіх різних комбінацій апаратних/програмних, і всі функції послідовно підтримуються;

Рисунок 1.6 – Тестування продуктивності

- тестування безпеки перевіряє програмне забезпечення на конфіденційність, цілісність, автентифікацію, доступність та недоторканність. Індивідуальні тести проводяться для запобігання будь-якому несанкціонованому доступу до коду програмного забезпечення.

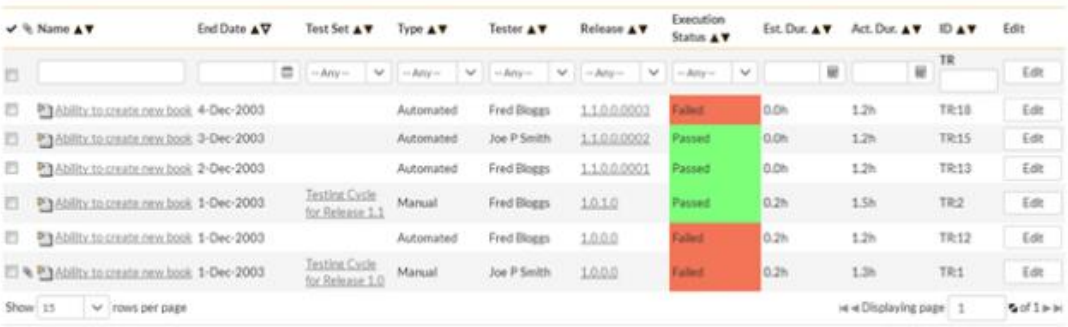

Рисунок 1.7 – Тестування сумісності

Юзабіліті-тестування розглядає аспект зручності використання програмного забезпечення кінцевого користувача. Простота, з якою користувач може отримати доступ до продукту, формує основну точку тестування. Юзабіліті-тестування розглядає п'ять аспектів тестування: навчання, ефективність, задоволеність, запам'ятовування та помилки.

Таким чином, сукупність розглянутих тестових випробувань може бути представлена у вигляді комплексного тестування (integration tests), що проводиться під час процесу інтеграції технічного та програмного забезпечення до валідації комп'ютерної системи з метою перевірки сумісності програмного забезпечення та технічного забезпечення комп'ютера.

Усі комплексні тести мають бути підготовлені на основі документації для користувача. Вони пишуться у формі сценаріїв, що представляють низку послідовних дій [20] користувачам і складаються з трьох основних компонентів:

- власне сценарію комплексного тестування із зазначенням дій, які мають бути досконалі під час виконання тесту;
- вхідних даних;
- очікуваних вихідних даних.

У комплексному тестуванні крім фахівців можуть брати участь і користувачі КІС, ґрунтуючись на одному з методів:

- досвідчена експлуатація, коли система тестується на робочому місці. Це дозволяє побачити КІС у роботі до того, як її почнуть експлуатувати;
- використання КІС у створенні виробника для внутрішніх потреб. Дозволяє усунути безліч помилок, але не дає змоги побачити деякі помилки інтеграції.

Таким чином, комплексне тестування КІС може бути процесом як контролю, так і випробування, при якому можна побачити її роботу в реальному середовищі користувача або в обстановці, яка створена, щоб максимально емулювати середовище користувача.

1.4 Види тестів під час виконання комплексного тестування КІС

Компонентами комплексного тесту є вихідні дані, документація, публікації для користувачів та сама система. Усі комплексні тести мають бути підготовлені з урахуванням публікацій для користувача (а чи не зовнішніх специфікацій). До зовнішніх специфікацій слід звертатися тільки для того, щоб розібратися в протиріччях між системою [16] та публікаціями про неї. Виділяють такі підвиди комплексного тестування:

1 тестування стресів або тестування з навантаженням - спроба піддати систему максимальному "тиску" (наприклад, спробу одночасно підключити до системи поділу часу 100 терміналів);

2 тестування обсягу – спроба пред'явити системі великі обсяги даних протягом тривалого часу, щоб визначити відповідність кількості даних і даних, зазначених у специфікації;

3 тестування конфігурації – спроба перевірити роботи мінімальної та максимальної конфігурації системи з будь-якою апаратною платформою або програмою, з якою КІС буде взаємодіяти;

4 тестування сумісності - спроба знайти несумісності нових та старих версій КІС, при цьому взаємодія користувачів з попередньою версією має повністю зберегтися;

5 тестування захисту [21] – спроба порушити таємність у системі;

6 тестування вимог до пам'яті – це спроба показати, що система не досягає тих цілей у пам'яті, які прописані в документації, що супроводжує;

7 тестування продуктивності – спроба продемонструвати, що дана програма не відповідає заявленим характеристикам, таким як час відгуку, рівень пропускної спроможності при певному навантаженні;

8 тестування процесів налаштування системи – спроба визначити якість та ергономічність роботи системи;

9 тестування надійності/готовності [22] – спроба довести, що система не задовольняє вихідним вимогам надійності (середній час між відмовами,

кількість помилок, здатність до виявлення, виправлення помилок та стійкість до помилок);

10 тестування засобів відновлення [18] – спроба перевірити здатність системи до відновлення (особливо у сфері роботи операційної системи, СУБД, систем передачі);

11 тестування зручності обслуговування – спроба проаналізувати очима обслуговуючого персоналу документи, що описують внутрішню логіку системи (вимоги до продукту, проекту, що визначають зручності обслуговування (супровід системи)), щоб зрозуміти, як швидко і точно вказати причину помилки, якщо відомі лише деякі її симптоми;

12 тестування публікацій – спроба перевірити точність усієї документації;

13 тестування психологічних чинників – спроба усунення дрібних недоліків, мінімізація незручності використання;

14 тестування зручності установки – спроба перевірити та усунути недоліки настановних процедур системи;

15 тестування зручності експлуатації – спроба вирішити проблеми ергономіки щодо взаємодії користувача із системою.

Аналіз видів комплексного тестування показав, що де вони зводяться до перевірки окремих функцій системи, а перевіряють працездатність КІС загалом. Найчастіше вони пишуться у формі сценаріїв, у яких вказуються дії, які здійснюються під час виконання тесту. Усі тестові сценарії мають бути методологічно та систематично опрацьовані. Під час тестування повинні бути використані вихідні дані та послідовності команд, які в документації користувача явно не рекомендуються або оголошуються забороненими. На рисунку 1.8 зображено сценарій комплексного тестування.

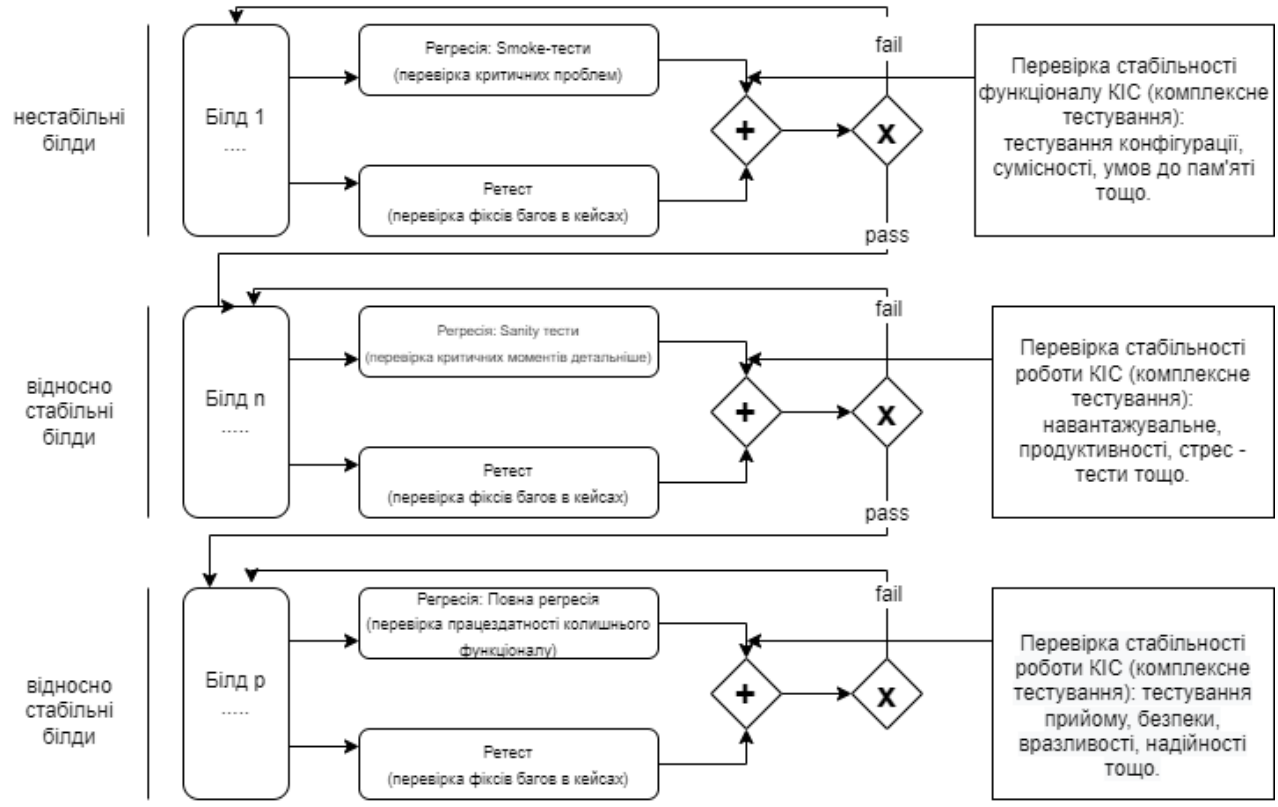

Рисунок 1.8 – Сценарій комплексного тестування

У ході аналізу теоретичних основ комплексного тестування корпоративних систем та аналізу його основних видів було зроблено висновок про те, що комплексне тестування КІС не зводиться до перевірки її функції, а орієнтовано на сценку якості та продуктивності. Для проведення комплексного тестування має бути розроблений сценарій, який має передбачити всі можливі варіанти, що забезпечують нестабільну роботу КІС.

2 АНАЛІЗ МЕТОДІВ ПРОВЕДЕННЯ КОМПЛЕКСНОГО ТЕСТУВАННЯ КІС

2.1 Основні критерії ефективності тестування КІС

Під тестуванням розуміють процес виконання програм на кінцевій множині вхідних даних X, отримання відгуку Y та його порівняння з еталонною безліччю вихідних значень $Y_{\text{вд}}$, з метою виявлення помилок та дефектів в ІС.

Пара $(x, y_{\text{вд}}): x \in X, y_{\text{вд}} \in Y_{\text{вд}}$ називається тестовим випадком (тест – кейс), а усі тестові випадки, сгруповані по певній ознаці, називаються тестоим комлпектом.

Рішення про наявність помилки у програмному забезпеченні (ПС) КІС приймається або при розбіжності результатів на одному з тестових випадків:

$$\exists_i, y^i_{\text{вд}} \in Y_{\text{вд}}, y^i \in Y: y^i_{\text{вд}} \neq y^i \qquad (2.1)$$

або, якщо відрізняються закони розподілу визідних даних.

В обох випадках вважається, що тестування пройшло успішно [23], оскільки виявлено як мінімум одну помилку.

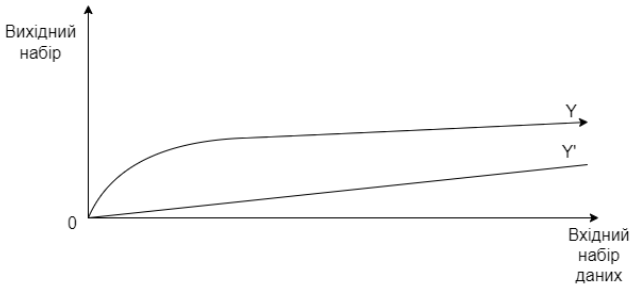

Рисунок 2.1 – Графіки залежностей результуючих та еталонних вихідних даних від вхідного комплекту

Інтенсивність виявлення помилок на одиницю витрат і надійність [24]

тісно пов'язані з часом тестування і тому з гарантією якості продукту (рисунок 2.2, блок А). Рух до зменшення кількості помилок, що залишилися, або до якості ІС призводить до використання різних способів тестування в процесі створення ПЗ. На рисунку 2.2 (блок В) наведено витратний компонент тестування [17] залежно від удосконалення інструментарію, що застосовується, та методів тестування.

На практиці відомі наступні способи тестування, упорядковані за їх застосуванням витрат:

- статичний спосіб тестування;
- модульний спосіб тестування;
- інтеграційний спосіб тестування;
- системний спосіб тестування;
- тестування реального оточення та в реальному часі.

Залежність ефективності впровадження перерахованих способів або їх можливості до виявлення відповідних класів помилок (рисунок 2.2, блок С) зіставлена на рисунку з витратами. Графік вказує, що з часом, у міру виявлення найбільш серйозних помилок та недоліків, ефективність низьковитратних методів падає разом з кількістю помилок, що виявляються.

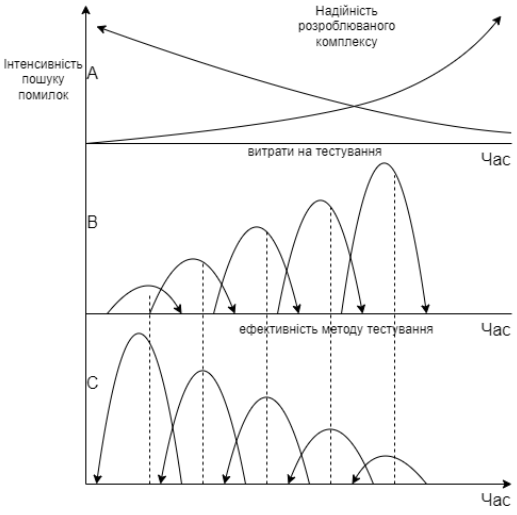

Рисунок 2.2 – Аналіз ефективності критеріїв тестування у процесі створення програмного продукту

Звідси випливає, що всі способи тестування не тільки мають право на існування, але й мають свою нішу, де вони добре виявляють помилки, тоді як за межами ніші їхня ефективність падає. Можна виділити вимоги до ідеального критерію тестування:

- достатній;
- повний;
- надійний;
- легко перевіряється.

При цьому існують такі класи критеріїв:

1 Клас Структурні критерії, що використовують модель програми у вигляді «білої скриньки», що передбачає знання вихідного тексту програми чи специфікації програми у вигляді потокового графа управління. До них відносяться:

- тестування команд (критерій C0);
- тестування гілок (критерій C1);
- тестування шляхів (критерій C2).

2 Клас Функціональні критерії [25], що використовують модель «чорної скриньки», що передбачає формулювання в описі вимог до програмного виробу та забезпечення контролю ступеня виконання вимог замовника в ПП. До них відносяться:

- тестування пунктів специфікації;
- тестування класів вхідних даних;
- тестування правил;
- тестування класів вихідних даних;
- тестування функцій;
- комбіновані критерії для програм та специфікацій.

3 Клас Критерії стохастичного тестування, які формуються в термінах перевірки наявності заданих властивостей у додатка, що тестується, коли набір детермінованих тестів (X, Y) має величезну потужність.

Критерії стохастичного тестування:

- статистичні методи закінчення тестування – стохастичні методи прийняття рішень про збіг гіпотез про розподіл випадкових величин (метод Стьюдента (St), метод Х-квадрат x^2 тощо);

- метод оцінки швидкості виявлення помилок [26] – заснований на моделі швидкості виявлення помилок, згідно з якою тестування припиняється, якщо оцінений інтервал часу між поточною помилкою та наступною занадто великий для фази тестування програми.

4 Клас Мутаційні критерії, орієнтовані на перевірку властивостей програмного виробу на основі підходу Монте-Карло, що дозволяє на основі дрібних помилок оцінити загальну кількість помилок, що залишилися у програмі.

Для розробки тестів слід вибрати методи, що реалізують критерії функціонального тестування, а також розглянути модель предметної галузі розробки.

2.2 Методи комплексного тестування КІС

Комплексне тестування КІС передбачає застосування методів тестування програмних продуктів та аналізу предметної галузі.

1 Виділимо серед методів тестування: методи функціональних діаграм та попарного тестування, а для аналізу предметної галузі розробимо онтологічну модель.

Функціональна діаграма є формальною мовою, якою транслюється специфікація програми, написана природною мовою. Побудова тестів цим методом здійснюється у кілька етапів:

- вхідні дані предметної області розбиваються на класи еквівалентності;
- за специфікацією визначаються причини та наслідки, при цьому під причиною розуміють окреме вхідне значення або клас еквівалентності вхідних даних, а наслідок – це вихідне значення або перетворення програми (дія, яку

вхідна умова чинить на стан програми);

- причини та наслідки перетворюється на булевський граф – це і є функціональна діаграма;

- діаграма додається примітками, що задають обмеження та описують комбінації причин та (або) наслідків, які є неможливими через синтаксичні або зовнішні обмеження;

- діаграма перетворюється на таблицю рішень з обмеженими входами, кожен стовпець якої відповідає тесту;

- стовпці таблиці рішень перетворюються на тести [27].

Під час розробки тестів часто доводиться аналізувати роботу системи з великою кількістю параметрів (наприклад, робота сайту різних браузерах).

Одним із підходів оптимізації кількості тестів є використання ортогональних масивів.

2 Ортогональний масив – це таблиця Lm(kn), де m – число рядків, n – число стовпців, яке відповідає числу вхідних параметрів, k – кількість варіантів значень елементів таблиці, та має такі властивості:

- будь-які два стовпці таблиці містять усі комбінації значень цих стовпців;

- якщо яка-небудь пара значень двох стовпців зустрічається кілька разів, всі можливі парні комбінації значень цих стовпців повинні зустрітися стільки ж раз.

Для тестування з використанням ортогональних масивів слід виконати такі кроки:

- встановити комбінації змінних для вхідних даних;

- визначити значення, які можуть приймати змінні;

- побудувати ортогональний масив, який має стовпець для кожної змінної;

- поставити у відповідність кожному тестовому випадку комбінацію значень змінних, розташованих у рядку побудованого масиву.

Протягом багатьох років було розроблено цілу низку комбінаторних

стратегій [28, 29] для того, щоб допомогти тестувальникам вибрати таку підмножину вхідних комбінацій, яка дозволила б максимально збільшити ймовірність виявлення дефектів: вибіркове тестування, «кожен-вибір» (each-choice) та «підстава вибору» (base choice), антирандомізація (antirandom), стратегія тестування t-способами (t-wise testing strategies) та інші. Парне тестування (pairwise testing) є найпомітнішим серед них. На рисунку 2.3 представлено залежність збільшення числа вичерпних та парних тестів від кількості тестових рівнів (можливої кількості значень параметрів).

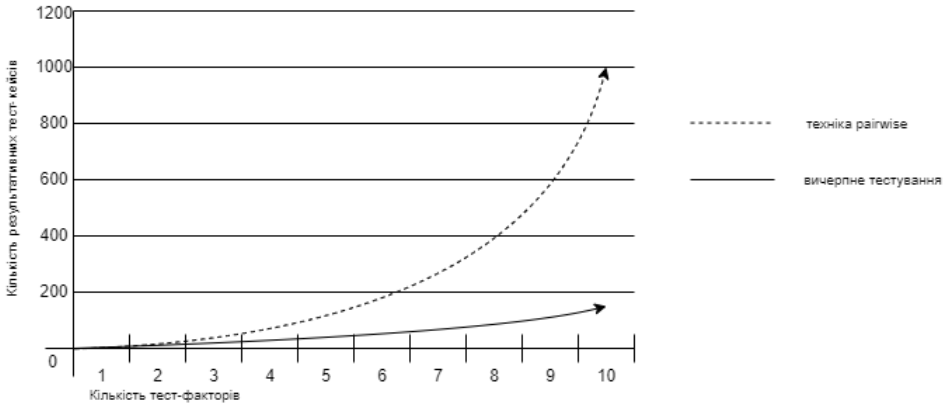

Рисунок 2.3 – Залежність збільшення числа вичерпних і парних тестів від кількості тестових рівнів

Парне тестування - метод проектування чорного ящика, в якому тестові випадки призначені для виконання всіх можливих дискретних комбінацій кожної пари вхідних параметрів.

Вихідні дані програмного забезпечення залежать від багатьох факторів, наприклад, від вхідних параметрів, змінного стану та конфігурації середовища. Парне тестування використовує: аналіз граничних значень і розподіл еквівалентності, які передбачають, що для кожної пари вхідних систем повинні існувати всі можливі параметри дискретних комбінацій цих параметрів.

Формально стратегія парного тестування визначається таким чином: маємо набір з N незалежних випробуваних факторів f1, f2, …, fN, де кожен

фактор f_i має L_i можливих рівнів $f_i = \{l_i, 1, \dots, l_i, L_i\}$, та набір тестів R. Кожен тест в R містить N тест-рівнів, по одному для кожного тест-фактору f_i, і, в сукупності, всі тести в R охоплюють всі можливі пари рівнів тест-факторів (відносяться до різних параметрів). Іншими словами, для кожної пари рівнів факторів l_i, p та l_j, q, де $1 \leq p \leq L_i$, де $1 \leq q \leq L_j$, та $i \neq j$ – існує по крайній одиниці тесту в R, який l_i, p та l_j, q .

Для аналізу предметної області розробки тестів застосовується онтологічний підхід. Під онтологією розуміється упорядкована тройка виду:

$$O = \langle C, R, F \rangle, \tag{2.2}$$

де:

C – кінцева множина концептів (понять, термінів) предметної області, яку представляє онтологія;

R – кінцева множина відносин між концептами заданої предметної області;

F – кінцева множина функцій інтерпретації (аксіоматизації), заданих на концептах та/або відносинах онтологій O.

Тестування слід здійснювати з погляду користувача [20], що передбачає повне розуміння того, навіщо система застосовуватиметься.

На рисунках 2.4, 2.5 представлені онтологічні моделі тестового випадку (класичне уявлення та реалізація у програмному середовищі).

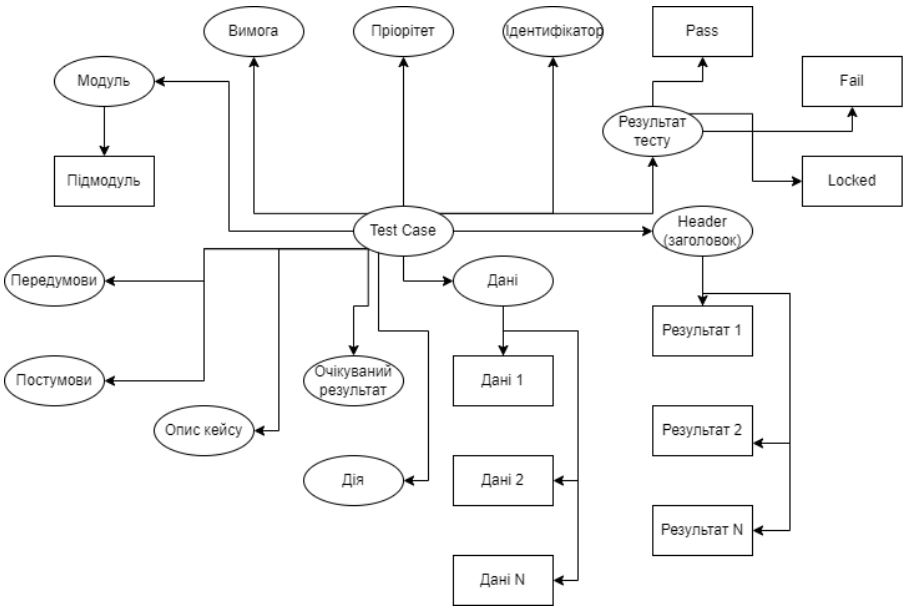

Рисунок 2.4 – Класична модель онтологічного випадку тест – кейсу

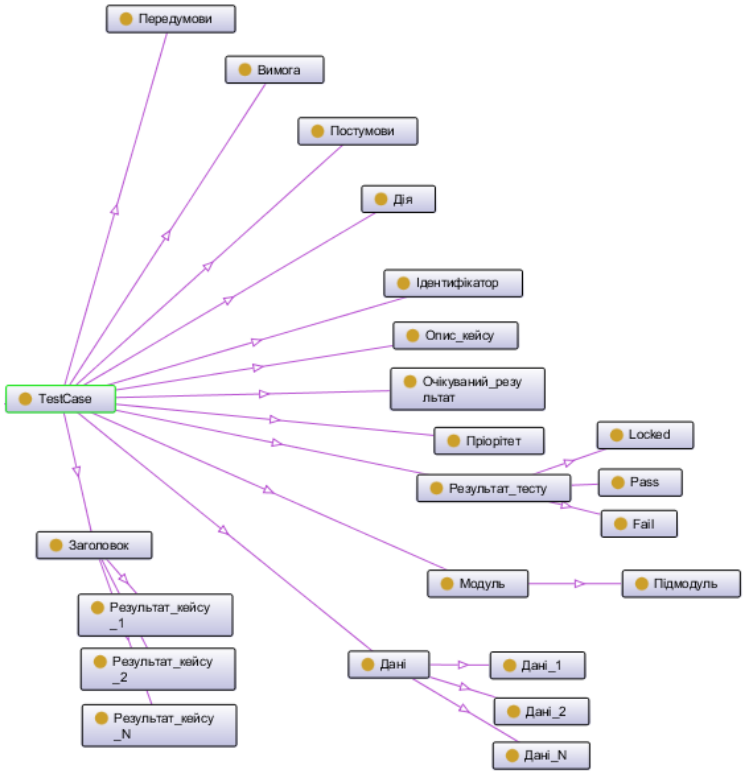

Рисунок 2.5 – Модель онтологічного випадку тест-кейсу за Protege

У зв'язку з тим, що сучасному класу КІС притаманний модульний принцип побудови, їх інтеграція [31], виділимо два підходи до комбінування

модулів: покрокове та монолітне тестування.

Перший підхід - монолітний метод, або метод «великого удару», що застосовується під час тестування та збирання програми.

Другий підхід - покроковий метод тестування чи складання, що передбачає, що модулі тестуються не ізольовано один від одного, а підключаються по черзі для виконання тесту до набору вже раніше відтестованих модулів до тих пір, поки до набору відтестованих модулів не буде підключений останній модуль.

Аналіз виділених методів модульного тестування КІС показав:

1 Монолітне тестування потребує великих витрат праці.

2 При монолітному тестуванні менша витрата машинного часу.

3 Монолітний метод надає великі можливості для паралельна організація роботи на початковій фазі тестування (тестування всіх модулів одночасно).

4 При покроковому тестуванні раніше виявляються помилки в інтерфейсах між модулями, оскільки раніше починається складання програми.

5 При покроковому тестуванні легше налагодження програм.

6 Результати покрокового тестування досконаліші і дають точніший аналіз щодо дефектів КІС.

Все це дозволяє зробити висновок, що покрокове тестування є кращим при розробці та апробації КІС.

Інтеграційне тестування спрямоване на перевірку взаємодії між частинами (модулями) програмного забезпечення КІС. Переконавшись у перевагах покрокового тестування перед монолітним, досліджуємо дві можливі стратегії тестування: низхідне та висхідне.

Існують два основні підходи [32] до проведення інтеграції: висхідна інтеграція (рисунок 2.6); низхідна інтеграція (рисунок 2.7).

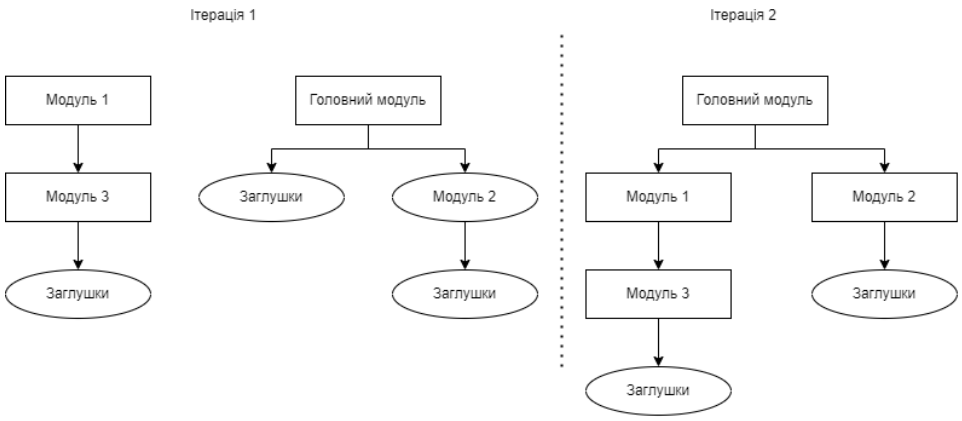

Рисунок 2.6 – Схема висхідної інтеграції

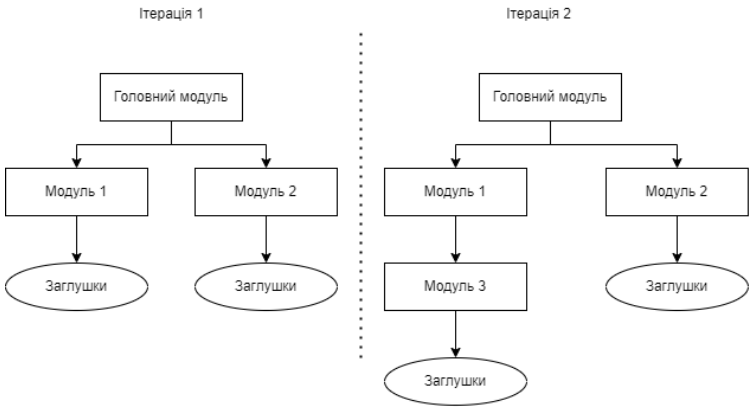

Рисунок 2.7 – Схема нисхідної інтеграції

Переваги та недоліки поданих підходів відображені в таблиці 2.1.

Таблиця 2.1 – Порівняння способів інтеграції

	Висхідна	Нисхідна
Переваги	Можливість ранньої перевірки коректності низькорівневої поведінки. Не потрібно написання заглушок, а визначити вимоги до входів/виходів модулів.	Можливість ранньої перевірки коректності високорівневої поведінки. Модулі можуть додаватися по одному незалежно один від одного. Не потрібно розробки безлічі драйверів.

Продовження таблиці 2.1

Недоліки	Відкладена перевірка високорівневої поведінки. Потрібна розробка драйверів.	Відкладена перевірка низькорівневої поведінки. Потрібна розробка «заглушок». Вкрай складно коректно сформулювати вимоги до входів/виходів часткової системи.

Для того щоб процес тестування КІС мав виправдану з економічної точки зору трудомісткість, необхідно заздалегідь виробити низку стратегій. Найбільш поширені:

- тестування методом «чорного» ящика - Тестування, кероване даними (data-driven testing) або тестування, кероване входом і виходом (input/output-driven testing);

- тестування методом «білого» ящика - Тестування, кероване логікою програми (logic-driving testing).

У таблиці 2.2 представлена основна різниця між тестуванням чорної скриньки та тестуванням білої скриньки.

Таблиця 2.2 – Порівняльний аналіз методів комплексного тестування

Тестування методом чорної скриньки	Тестування методом білої скриньки
Це підхід до тестування, який використовується для тестування програмного забезпечення без відома внутрішньої структури програми чи програми.	Це підхід до тестування, при якому тестувальник знає внутрішню структуру.
Тестування ґрунтується на зовнішніх очікуваннях; внутрішня поведінка програми невідома.	Внутрішня робота відома, і тестер може тестувати відповідно.

Продовження таблиці 2.2

Цей тип тестування ідеально підходить для вищих рівнів тестування, таких як тестування системи, прийомне тестування.	Тестування найкраще підходить для нижчого рівня тестування, як модульне тестування, інтеграційне тестування.
Знання з програмування не потрібні для проведення тестування Black Box.	Для тестування White Box необхідні знання з програмування.
Знання впровадження не вимагають проведення тестування Black Box.	Повне розуміння потребує впровадження тестування WhiteBox.
Основна мета цього тестування - перевірити, яку функціональність перевіряється системи.	Основною метою тестування White Box є перевірка якості коду.
Тестування може розпочатися після підготовки документа специфікації вимог.	Випробування можна розпочати після підготовки до документації з детального проектування.
Виконується кінцевим користувачем, розробником та тестувальником.	Зазвичай це робиться тестером та розробниками.
Базується на методі спроб і помилок.	Домен даних та внутрішні межі можуть бути перевірені.
Час менш вичерпний і трудомісткий.	Час вичерпний і трудомісткий метод.
Добре підходить і ефективний для великих сегментів коду.	Це дозволяє видалити зайві рядки коду, що може призвести до прихованих дефектів.
Оновлення сценарію тестування автоматизації має важливе значення, якщо ви часто змінюєте програму.	Автоматизовані тестові випадки можуть стати марними, якщо база коду швидко змінюється.

Для комплексного тестування КІС також застосовуються наступні методи:

Альфа-тестування – це ручне тестування потенційними користувачами, замовниками чи незалежною командою тестування на стенді розробки. Альфа-тестування часто використовується як форма внутрішнього приймального тестування перед проведенням тестування бета.

Альфа-тестування дозволяє фільтрувати, уточнювати і передавати розробникам дефекти, що надходять, з докладним описом, що значно скорочує час, а також дозволяє скорочувати трудовитрати розробників на пошук причини дефекту і його виправлення.

У межах проведення альфа-тестування вирішуються такі:

- підготовка розкладу тестування;
- організація учасників тестування;
- відбір і уточнення зауважень, що надходять;
- реєстрація дефектів.

Бета-тестування проводиться після альфа-тестування та може використовуватись як приймальне тестування зовнішніми користувачами. Бета-версія системи передається групі користувачів поза командою розробки [34], щоб зменшити кількість дефектів. Іноді версія передається командам, щоб отримати зворотний зв'язок від якнайбільшої кількості майбутніх користувачів.

Ключові переваги:

- отримання відгуків та побажань від потенційних користувачів КІС;
- підвищення якості проведеного тестування у задані терміни та ліквідація проблем, пов'язаних із тестовим середовищем.

Таким чином, використання разом розглянутих методів сприяє більш якісному проведенню комплектного тестування КІС.

2.3 Комплексне тестування КІС на її продуктивність

Сучасні корпоративні інформаційні системи часто змушені працювати з

максимальним навантаженням. Збої і помилки, що виявляються при цьому, ведуть до збитків, які часто неможливо оцінити. Тестування продуктивності дозволяє мінімізувати ці ризики та вирішити цілу низку пов'язаних з ними проблем.

Тестування продуктивності дозволяє оцінювати та планувати продуктивність КІС, а також керувати нею в умовах планового [33], підвищеного та пікового навантаження. Навантажувальне тестування виконується з метою оцінки поведінки системи за нормальних умов і за прогнозованому піку навантаження. Стресове тестування передбачає ретельну перевірку в екстремальних умовах з метою оцінки стабільності КІС.

2.3.1 Тестування продуктивності КІС

Тестування продуктивності (Performance Testing) – це тестування, яке виконується для визначення того, як компоненти системи працюють у конкретній ситуації. Основна мета тестування продуктивності включає встановлення еталонної поведінки системи.

Тестування продуктивності не спрямовано на пошук дефектів у додатку. Успішний тест продуктивності має спроектувати більшість проблем продуктивності, які можуть бути пов'язані з базою даних, мережею, програмним забезпеченням, обладнанням тощо. Тестування продуктивності проводиться з метою надання заінтересованим сторонам інформації про їх застосування, що стосується швидкості, стабільності та масштабованості. Тестування продуктивності визначає, чи програмне забезпечення ІС відповідає вимогам швидкості, масштабованості та стабільності при очікуваних робочих навантаженнях. Тестування продуктивності проводиться, щоб переконатися, що програма працює досить швидко, щоб утримувати увагу та інтерес користувача. Тести продуктивності можуть вказувати на потенційний вплив змін на продуктивність, особливо після внесення змін до технології, реалізації

або конфігурації.

Нижче наведено загальний процес тестування продуктивності:

1 Визначити своє середовище тестування – ознайомитися з деталями апаратного, програмного забезпечення та мережевих конфігурацій, використаних під час тестування.

2 Визначити критерії прийнятності продуктивності – сформулювати цілі та обмеження пропускної спроможності, часу відгуку та розподілу ресурсів, критерії успіху проекту поза цими цілями та обмеженнями.

3 Планувати та проектувати тести продуктивності – визначити, як використання може змінюватись серед кінцевих користувачів, та визначити ключові сценарії для тестування у всіх можливих випадках використання.

4 Налаштування середовища тестування – підготувати середовище тестування перед виконанням, організувати інструменти та інші ресурси.

5 Реалізувати тестовий дизайн – створити тести продуктивності відповідно до тестового дизайну.

6 Запустити тести – виконати та оцінити результати тесту.

7 Аналізувати, налаштувати та повторно протестувати, поширити результи випробувань; виконати точне налаштування та перевірити знову, щоб побачити, чи є покращення чи зниження продуктивності.

Створювані сценарії тестування продуктивності мають бути близькими до реальних умов. Технологія тестування продуктивності має підтримувати сценарії, які збільшують кількість користувачів, а й імітують їх поведінка. Типовою поведінкою може бути, наприклад, відвідування користувачем домашньої сторінки, вхід до системи, перехід за посиланням на статтю, додавання товару до кошика та здійснення покупки.

У наступному фрагменті коду показаний опис простого моделювання для Gatling на Scala (рисунок 2.8).

```
import io.getling.core.Predef/_
import io.getling.core.structure.ScenarioBuilder
import io.getling.http.Predef/_
import io.getling.http.protocol.HttpProtocolBuildder
import scala.concurrent.duration._

class CarCreationSimulation extends Simulation {
    val httpConf: HttpProtocolBuildder = http
    .baseURL(http://test.car-manufacture.example.com/car-
manufacture/resources)
        .acceptHeader("*/*")

    val scn: ScenarioBuilder = scenario("create_car")
        .exec(http('request_1")
            .get("/cars"))
        .exec(http('request_1")
            .post("/cars"))
            .body(StringBody("""{"id": "X123A234", "color": "RED",
                        "engine": "DIESEL"}""")).asJSON
            .check(header("Location").saveAs("locationHeader")))
        .exec(http('request_1")
            .get("${locationHeader}"))

    Pause(1 second)

Detup(
    scn.inject(rampUsersperSec(10).to(20).during(10 second))
        ).protocols(httpConf)
        .constantPauses
}
```

Рисунок 2.8 – Фрагмент опису обробки клієнтських запитів

Сценарій create_car включає три клієнтські запити, які читають список всіх автомобілів, створюють автомобіль і підключаються до створеного ресурсу. Сценарії настроюють кілька віртуальних користувачів. Кількість користувачів починається з 10 і збільшується до 20 протягом 10 секунд. Тестування продуктивності [34] дозволяє визначити максимальну інтенсивність операцій, коли він система задовольняє вимогам на час відгуку. На ринку є безліч інструментів для тестування продуктивності:

- NeoLoad – це платформа для тестування продуктивності, розроблена для DevOps, яка легко інтегрується у існуючий конвеєр безперервної доставки. З NeoLoad команди тестують у 10 разів швидше, ніж з традиційними інструментами, щоб відповідати новому рівню вимог протягом усього життєвого циклу розробки програмного забезпечення Agile – від компонентів до повних тестів навантаження в масштабі всієї системи;

- LoadView Testing – платформа для тестування інфраструктури у будь-якому масштабі. LoadView пропонує навантажувальне тестування на основі

хмарних обчислень на вимогу;

- HP LoadRunner – це найпопулярніший на сьогоднішній день інструмент для тестування продуктивності. Цей інструмент здатний моделювати сотні тисяч користувачів, піддаючи додатки реальному навантаженню, щоб визначити їхню поведінку при очікуваному навантаженні. Loadrunner має віртуальний генератор користувачів, що імітує дії живих користувачів;
- Jmeter – один з провідних інструментів, що використовуються для тестування навантаження веб-серверів і серверів додатків [40].

Тестування продуктивності рекомендується проводити для нової версії програмного забезпечення, що планується до впровадження. Воно дозволяє виявити та запобігти відмовим КІС перед впровадженням у промислову експлуатацію; визначити максимальну кількість одночасно працюючих користувачів у системі та максимальну кількість одночасно виконуваних операцій без втрати якості обслуговування.

2.3.2 Навантажувальне тестування КІС

Навантажувальне тестування (load testing) [35] – даний тип тестування дозволяє оцінити поведінку системи при зростаючому навантаженні. Метою тесту навантаження є визначення максимального навантаження, яке може витримати система.

У ролі навантаження може бути кількість користувачів, а також кількість операцій на сервері. Продуктивність при цьому визначається такими факторами:

- швидкістю роботи програмного забезпечення;
- швидкістю роботи апаратного забезпечення;
- швидкістю роботи мережі.

Під час тестування можуть здійснюватися такі операції, що дозволяють більш точно виміряти продуктивність та визначити «вузьке місце» системи:

- вимірювання часу виконання обраних операцій при певній

інтенсивності виконання цих операцій;

- визначення кількості користувачів, що одночасно працюють з додатком;

- визначення меж прийнятної продуктивності зі збільшенням навантаження (у разі збільшення інтенсивності виконання цих операцій).

Приклад тесту навантаження показаний на рисунку 2.9. Цей тест проводиться визначення кількості користувачів, із якими може працювати система.

У цьому тесті 100 користувачів додаються через кожні 30 секунд, доки навантаження не досягне 1000 користувачів. Кожен крок займає 30 секунд, і Jmeter чекає 30 секунд [44]rt , перш ніж розпочати наступний крок. Як тільки навантаження досягне 1000 потоків, всі вони продовжуватимуть працювати протягом 300 секунд (5 хвилин) разом, а потім нарешті зупиняють 10 потоків кожні 3 секунди.

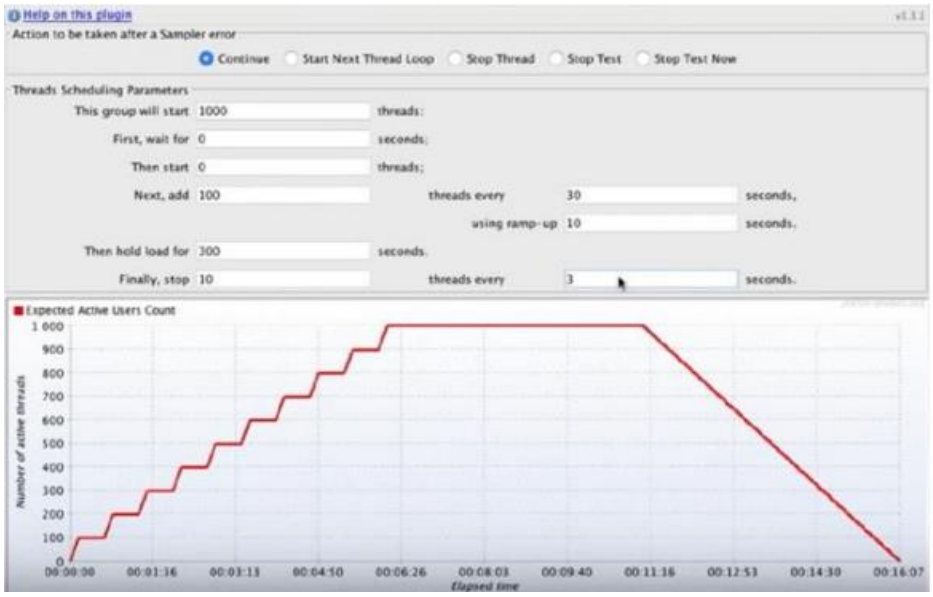

Рисунок 2.9 – Приклад виконання тесту навантаження

Інший приклад тесту навантаження показаний на рисунку 2.10 де зображено тест на піки при раптовому збільшенні кількості користувачів до 7000.

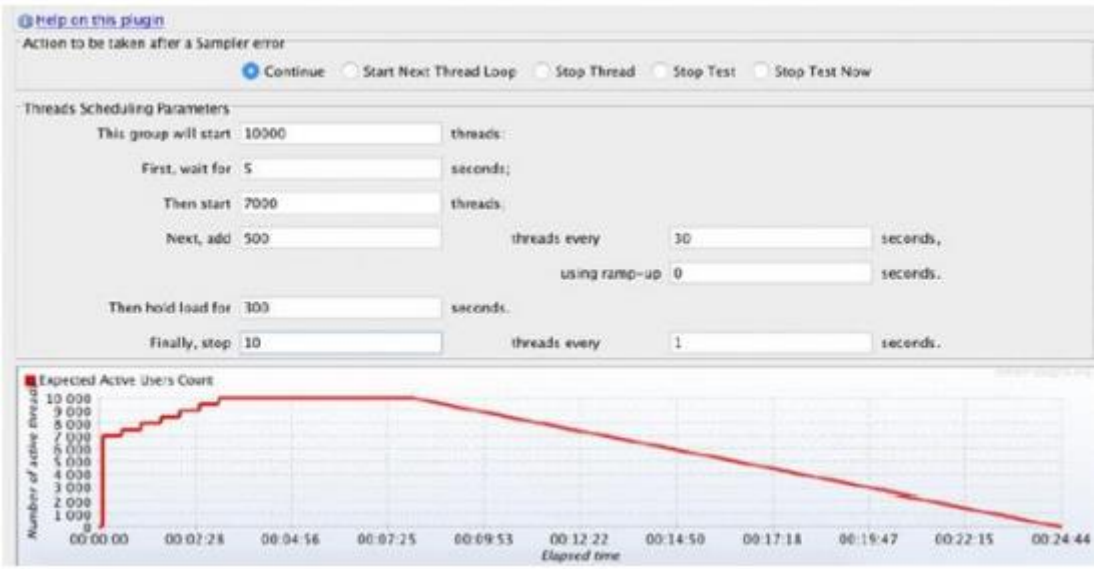

Рисунок 2.10 – Приклад виконання тесту навантаження

Навантажувальне тестування – дослідження можливості застосування зберігати задані показники якості при навантаженні в допустимих межах і деякому перевищенні цих меж (визначення «запасу міцності»).

2.3.3 Стрес-тестування КІС

При стрес-тестуванні [36] виконуються різні дії з перевантаження існуючих ресурсів надмірними завданнями спробі зламати систему. Негативне тестування, яке включає видалення компонентів із системи, також проводиться у рамках стрес-тестування. Це тестування, також відоме як тестування втоми, повинно відображати стабільність додатка шляхом його тестування за межами пропускної здатності.

Мета стрес-тестування полягає в тому, щоб встановити відмову системи та відстежити, як система коректно відновлюється [37]. Завдання тут полягає в тому, щоб налаштувати контрольоване середовище перед запуском тесту, щоб ви могли точно фіксувати поведінку системи за непередбачуваних сценаріїв.

Мета стрес-тестування полягає в тому, щоб проаналізувати звіти після аварії, щоб визначити поведінку програми після збою. Найбільша проблема

полягає в тому, щоб гарантувати, що система не ставить під загрозу безпеку конфіденційних даних після збою. При успішному стрес-тестуванні система повернеться у нормальний стан разом із усіма її компонентами навіть після найстрашнішого збою.

Підсумовуючи, у таблиці 2.3 представлені основні різницю між цими трьома типами тестування.

Таблиця 2.3 – Порівняльний аналіз методів тестування продуктивності

	Тестування продуктивності	Тест навантаження	Стрес - тест
Домен	Суперсет навантажувального та стрес-тестування	Підмножина тестування продуктивності	Підмножина тестування продуктивності
Об’єм	Дуже широка сфера застосування. Включає безліч типів тестування, пов'язаних з перевіркою на обсяг, навантаження та витривалість.	Вужче охоплення в порівнянні з тестуванням продуктивності. Включає об'ємні випробування та випробування на витривалість.	Вужче охоплення в порівнянні з тестуванням продуктивності. Включає тестування на всмоктування.
Основна ціль	Щоб встановити стандарти для програми.	Щоб визначити верхню межу системи	Визначити, як система поводиться при інтенсивних навантаженнях та як відновлюється.

Продовження таблиці 2.3

Межа навантаження	Обидва – нижче та вище меж перерви.	До межі розриву	Вище межі розриву
Вивчення атрибутів	Використання ресурсів, надійність, масштабованість, час відгуку, пропускна спроможність тощо.	Пікова продуктивність, пропускна здатність сервера, час відгуку при різних рівнях навантаження (нижче за межу)	Стабільність за межами пропускної спроможності, час відгуку (вище за межу розриву)
Проблеми, що виявлені за допомогою цього типу тестування	Усі помилки продуктивності, включаючи роздмухування під час виконання, можливості оптимізації, проблеми, пов'язані зі швидкістю, затримкою, пропускною здатністю тощо.	Проблеми з балансуванням навантаження, проблеми з пропускною спроможністю, проблеми з пропускною спроможністю системи, поганий час відгуку, проблеми з пропускною спроможністю тощо	Пробіли в безпеці з перевантаженням, проблеми з пошкодженням даних у ситуації навантаження, повільність, витоку пам'яті тощо

У рамках тестування продуктивності збирається статистика використання системи, на основі якої складається профіль навантаження. Після цього обчислюється початкова точка та розмір кроку для збільшення інтенсивності виконання операцій.

У ході виконання тестування визначається максимальна продуктивність системи [38]: найбільша інтенсивність виконання операцій з необхідною якістю обслуговування (SLA); пікова продуктивність системи, коли відбувається погіршення показників якості обслуговування операцій (час виконання, відмови).

2.4 Методи та сценарій приймального тестування КІС

Приймальний тест – це комплексне тестування, необхідне для визначення рівня готовності системи до подальшої експлуатації.

Тестування проводиться на підставі набору тестових сценаріїв, що охоплюють основні бізнес-операції системи [39]. Ключові переваги:

- дозволяє виявити системні порушення;
- дозволяє виявити дефекти, пов'язані із зручністю та простотою використання;
- залучення досвідчених компетентних спеціалістів дозволяє грамотно, якісно та в задані терміни провести процес приймання тестування.

Основні етапи приймального тестування:

1 підготовка – включає розробку програми та методики випробувань та підготовку приймальних тестів;

2 проведення – супровід клієнта під час проведення приймальних тестів (заклад дефектів, відстеження коректності та швидкості виконання тестування);

3 звіт – клієнту надається докладний звіт із переліком помилок, які потрібно усунути перед запуском системи в експлуатацію.

Напрями приймального тестування:

- операційне тестування – перевірка системи на здатність виконувати

свою роль у середовищі експлуатації згідно бізнес-моделі;

- альфа-тестування – перевірка незалежною командою тестування;

- тестування користувача - перевірка придатності системи для впровадження кінцевими користувачами;

- бета-тестування – тестування зовнішніми користувачами, потенційними клієнтами.

Сценарій приймального тестування має містити чітко сформульовані розділи, що представлені на рисунку 2.11.

Рисунок 2.11 – Сценарій тестування прийому

Найбільш повним та різнобічним приймальним випробуванням має піддаватися перша базова версія КІС. Комплексне тестування ПЗ КІС гарантує перевірку виконання всіх вимог технічного завдання (ТЗ). Запропонований сценарій проведення приймального тестування дозволяє фіксувати умови неправильної роботи програм та характер прояву дефектів.

При завершальних випробуваннях основна увага, крім перевірок функціональної придатності, має зосереджуватись на підготовці стресових тестів, тестуванні в режимах граничного використання ресурсів, надійності функціонування КІС. Для цього необхідно проводити випробування у два послідовні етапи – Альфа- та Бета-тестування.

3 МЕТОДИ ІНТЕГРОВАНОГО ТЕСТУВАННЯ КОРПОРАТИВНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Тестування навантаження, засноване на моделях

Розглянемо підхід, заснований на моделях стосовно тестування навантаження КІС, в межах якого застосовуване програмне забезпечення (програмне забезпечення) тестується в ІТ - середовищі функціонування, включаючи СУБД, базу даних, системне програмне забезпечення, розгорнуте на обладнанні технічної системи ІС.

З навантаженням тестування КІС у процесі їх роботи головна мета - не перевірити правильність функціонування прикладних програм (передбачається, що функціональне тестування проводиться), а оцінка оперативних характеристик інформації Система, яка включає прикладне програмне забезпечення. Зазвичай це так звані "цільові показники" або нефункціональні вимоги, тобто пропускна здатність продуктивності ПП, реактивність у вирішенні функціональних проблем та ряду інших.

З формалізацією предметної області (для класу систем) утворюються початкові дані:

- інформація про вибраний тип тестування навантаження (оцінювальна, аналітична, настрої, регресія);
- інформація про вимірювані характеристики та показники виконання;
- інформація про структуру системи з точки зору варіантів подачі методів навантаження та вимірювання;
- інформація про заплановане навантаження в структурованій формі.

Уявіть наступні моделі, за допомогою яких при встановленні завдання експерименту навантаження необхідні характеристики, показники та вимірювані величини, що належним чином характеризують процес функціонування перевіреної інформаційної системи (рисунок 3.1):

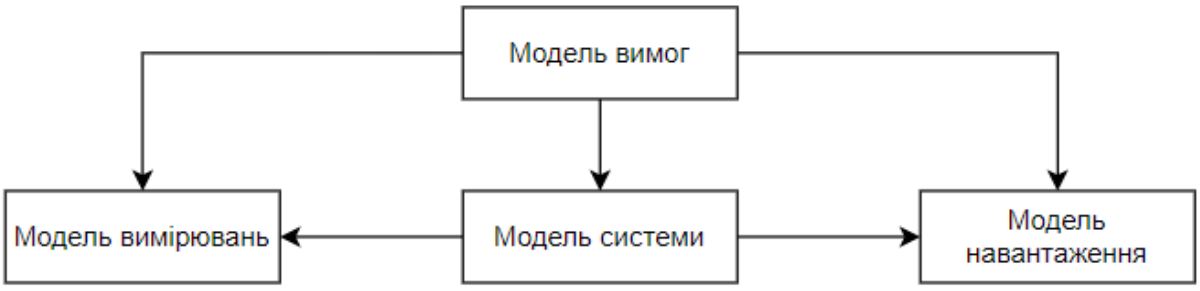

Рисунок 3.1 – Зв’язок моделей

1 Модель вимог

Характеризує тип тестової системи, склад нефункціональних вимог (права бізнесу та технічні вимоги) перевіреної інформаційної системи. Модель вимог може бути представлена у наступній формі:

$$R = B \cup T, \tag{3.1}$$

де R – багато вимог до системи;

- B – безліч ділових прав, пов'язаних з технологічними процесами, корпоративними нормами, політиками, стандартами, законодавчими актами, внутрішньокорпоративними ініціативами, бухгалтерською практикою, алгоритми розрахунку тощо;

- T – Багато технічних вимог, які встановлюють технічні властивості, які система повинна мати, наприклад, характеристики продуктивності, надійності та доступності.

Модель вимог – це словесний опис нефункціональних вимог, згрупованих за типами та об'єктами, до яких вони застосовуються. Ця модель також містить критерії оцінки характеристик, отриманих в результаті навантаження та обчислених показників.

2 Модель системи

Системна модель описує структуру системи як мережу систем масових

служб (включаючи склад типу "ресурсу").

Системна модель може бути представлена у такій формі:

$$\Sigma = \{U(p)\}, \{S\}, K_s, K_u, \quad (3.2)$$

де $U(p)$ – багато пристроїв тестувального об'єкта з характеристиками продуктивність p;

- S – багато програмних систем (компоненти, послуги);
- K_s – матриця з'єднань між програмними системами, лінії яких є джерелами, стовпчики - це приймачі, а комірки вказують на наявність зв'язку між ними;
- K_u – матриця з'єднань програмних систем із пристроями, характеризуючи кількість ресурсів, виділених пристроєм для програмного комплексу. Рядки цієї матриці - це програмні системи, стовпці - це обчислювальні комплекси. Елементи матриці є векторами виділених ресурсів.

Матриці K_s та K_U для представленої схеми виглядають наступним чином:

$$K_s = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}, \; K_U = \begin{pmatrix} p_1 & 0 & 0 & 0 \\ 0 & p_2 & 0 & 0 \\ 0 & 0 & p_3 & 0 \\ 0 & 0 & 0 & p_4 \\ 0 & 0 & 0 & p_4 \end{pmatrix}, \quad (3.3)$$

де p_i – характеристики продуктивності вектор, що характеризує кількість ресурсів, виділених для визначеного комплексу програмного забезпечення.

Системна модель – це опис об'єкта тестування, який включає програмне забезпечення та технічні засоби, їх з'єднання та виділені ресурси (пам'ять, процесор тощо). На рисунку 3.2 представлений приклад системи системи у вигляді графічної схеми взаємодії програмного забезпечення та обчислювальних комплексів.

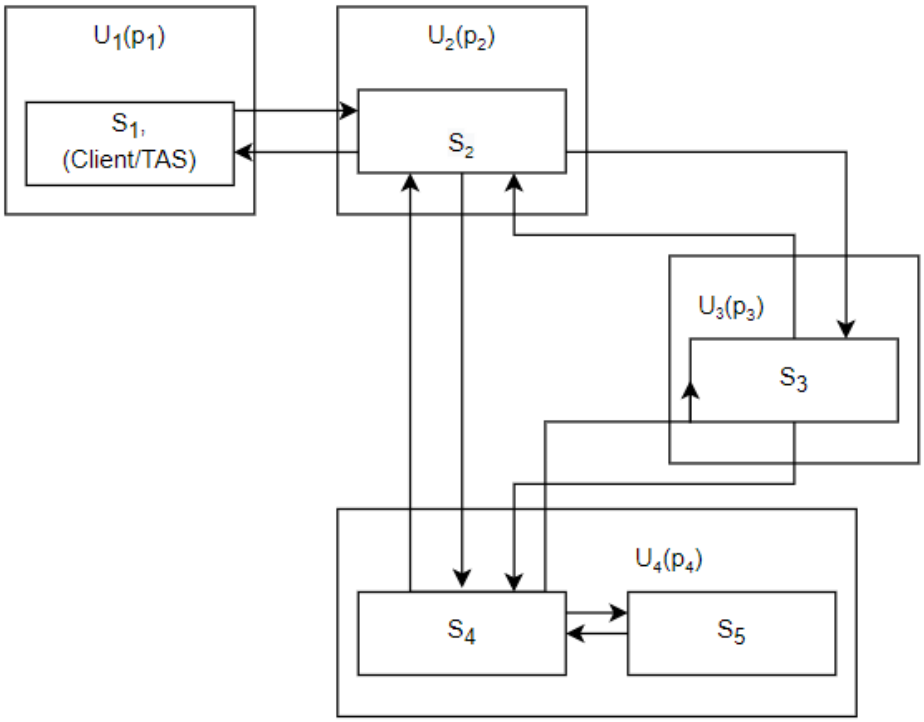

Рисунок 3.2 – Приклад системної моделі у вигляді графічної схеми

3 Модель навантаження

Це опис кількості та типів вимог до послуг для системи, закону розподілу вимог до обслуговування в часи експерименту, правила отримання вимог до обслуговування в системі, точки введення вимог до послуги в систему .

Модель навантаження може бути представлена у наступній формі:

$$L = \{F, M, I\}, \tag{3.3}$$

де F – багато функцій, що характеризують розподіл навантаження, введеного в систему;

- M – багатовимірна матриця, розміри яких можуть бути типом навантаження: джерела потоків навантаження, назви та типи потоків та елементи з кількісними характеристиками;

- I – багато інтерфейсів для введення навантаження (у вигляді посилання на модель системи).

Потік навантаження структурується відповідно до його динамічних (F) та статичних (м) властивостей.

Модель навантаження - це опис даних тестів та правила їх введення в систему. Насправді, відповідно до моделі навантаження, початкові дані утворюються для отримання інструментів завантаження.

4 Модель вимірювання

Модель вимірювання - визначає склад зібраних характеристик, показників та значень, інтерфейс для впровадження вимог до системи, методу їх алгоритмів збору та перетворення, а також критерії оцінювання результатів.

Модель вимірювання розроблена для об'єднання опису:

- способи отримання виміряних значень із навантаженням тестування ІС;
- основні можливості встановлення процесу вимірювання;
- типові способи оцінки показників та характеристик;
- загальні властивості інструментів для аналізу можливостей їх використання для автоматизації вимірювань.

Модель вимірювання може бути представлена у наступній формі:

$$\Delta = \{|U|\}, \tau, \mu, R, \omega, \qquad (3.4)$$

де $\{|U|\}$ - список вимірюваних значень для кожного типу інформаційної системи або частини пристроїв;

- τ - частота та обертання вимірювань;
- μ - багато обчислених показників та їх взаємозв'язок з вимірюваними значеннями;
- R - типові правила та алгоритми отримання обчислених показників;
- ω - типові критерії оцінки результатів.

Для планування експерименту з навантаженням ключова роль відіграється шляхом вибору переліку вимірюваних характеристик продуктивності (таблиця 3.1), оскільки висновки роблять на основі отриманих значень цих характеристик щодо результатів експерименту.

Таблиця 3.1 – Типи характеристик продуктивності

Характеристики продуктивності	Показники обчислення
Реактивність	Середній час відгуку
	Середній час очікування
	Середній час обслуговування
Продуктивність	Пропускна здатність
	Виробництво
Використання	Утилізація ресурсу
	Відносна пропускна здатність

Реактивність важлива для КІС, оперативні характеристики ІС можуть бути часом відповіді (реакції) системи на запит користувача або час очікування вирішення проблеми (наприклад, проблема мінімізації може бути вирішена часом очікування впровадження бізнес - транзакцій).

Характеристики реакційної здатності визначаються як час між представленням системи вхідних даних та поява відповідних вихідних даних. Реактивність оцінюється або обчислюється на основі таких показників ефективності:

- середній час відгуку;
- середній час обслуговування;
- середній час очікування.

Час відповіді обчислюється як період між початком транзакції та завершенням останнього етапу транзакції:

$$T_r = T_s + T_w, \tag{3.5}$$

де T_r – час відповіді, який розвивається з моменту служби;

- T_s – час, за який виконується обробка;
- T_w – час очікування (чекає на ресурс).

Індикатори реактивності залежать від типу системи, структури її вхідних

та вихідних точок.

Для КІС важлива їх інтегральна пропускна здатність, оцінена як кількість ділових операцій, оброблених системою на одиницю часу (продуктивність).

При оцінці продуктивності використовуються, як правило, безпосередньо вимірювані або обчислені значення наступних показників ефективності:

- виробництво (V);
- пропускна здатність або абсолютна пропускна здатність (C).

Виробництво (V) визначається як кількість завершених операцій протягом певного періоду часу:

$$V_i = \frac{N_i}{T}, \tag{3.6}$$

де i = 1, ..., n – порядковий номер періоду часу;

- N_i – кількість завершених транзакцій;
- T – період часу.

Пропускна здатність (C) – максимально можлива кількість завершених транзакцій за одиницю часу:

$$\mathrm{C} = \max(V_1 \dots, V_n), \tag{3.7}$$

Характеристики використання призначені для того, щоб описати, якою мірою використовуються ресурси системи, що тестується при заданому навантаженні.

До використання належать такі показники продуктивності:

- утилізація ресурсу (коефіцієнт використання ресурсу);
- відносна пропускна здатність.

Утилізація ресурсу показує, яку частину часу ресурс використовується у обслуговуванні транзакцій. Загальна формула:

$$U = \frac{\sum_{i=1}^{n} t_{si}}{T} \text{ x } 100\%, \tag{3.8}$$

де n – кількість обслужених транзакцій;

- t_{si} – час обслуговування і-ої транзакції;
- T – період обслуговування.

Характеристики використання ресурсів [41] та їх кількість суттєво залежать від використовуваного в системі обладнання та ресурсів, що входять до його складу: процесорів, оперативної пам'яті, зовнішніх носіїв, каналів введення-виводу тощо.

Технологія тестування навантаження, заснована на моделях (рисунок 3.3), складається з наступних етапів:

- визначення цілей тестування – за допомогою моделі вимог відбувається визначення правил формалізації вимог до експлуатаційних характеристик системи та формується модель вимог;
- розробка програми та методики випробувань - модель навантаження використовується для визначення можливої статичної та динамічної структури та складу потоку навантаження;
- підготовка до тестування – на основі моделі навантаження виконується налаштування засобів планування тестування та генератора тестових даних;
- подача навантаження – засоби автоматизації виконують процедури подачі навантаження в точки входу, задані для кожного типу навантаження моделі системи;
- збір даних – виконується автоматизований збір значень вимірюваних характеристик;
- інтерпретація та аналіз результатів – засобами автоматизації [42] використовуються всі чотири моделі: модель вимог для зіставлення результатів експерименту з вимогами до системи; модель навантаження та модель системи забезпечують формування звіту про умови проведення експерименту.

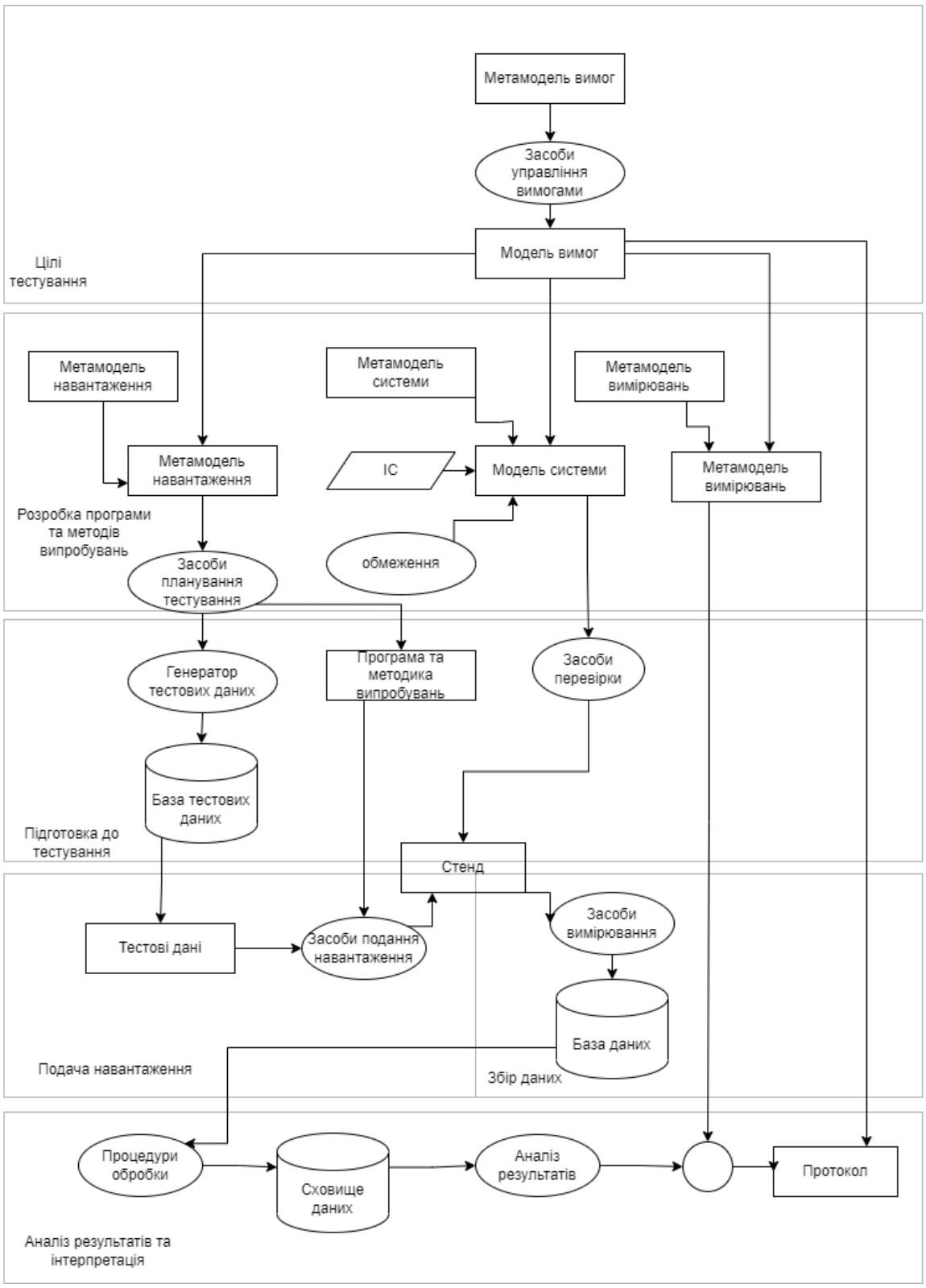

Рисунок 3.3 – Процес тестування навантаження з використанням моделей

Технологія, заснована на використанні моделей, що описують вимоги до

проведення експерименту навантаження і властивості об'єкта тестування, охоплює всі аспекти планування, проведення експерименту навантаження і аналізу його результатів.

Використання моделей суттєво (у кілька разів) знижує трудомісткість навантажувального тестування, забезпечує можливість повторного використання підготовленого експерименту, наприклад, при контролі деградації інформаційної системи в її життєвому циклі.

3.2 Сценарій тестування навантаження з використанням інструментальних засобів

Розглянемо тестування навантаження [43] для оцінки продуктивності простого HTTP-запиту GET з 20 потоків з 100 000 ітерацій.

Сторона сервера (додаток, що тестується): Процесор: 4x Xeon L5520 2,27 ГГц; RAM: 8 ГБ; ОС: Microsoft Windows Server 2008 R2 x64; Сервер програм: IIS 7.5.7600.16385.

Клієнтська сторона (генератор навантаження): Процесор: AMD Ryzen 7 4700 2.0 – 4.1 ГГц; RAM: 40 ГБ; ОС: Ubuntu 20.04 LTS 64-бітна.

Для проведення тестування будемо використовувати інструментальні засоби [19], кожен інструмент надсилатиме запити так швидко, як може:

1 Grinder – це безкоштовне середовище навантажувального тестування на основі Java, доступне за ліцензією BSD у стилі open-source. На рисунку 3.4 представлений фрагмент проведення тесту навантаження, виконаного з використанням Grinder;

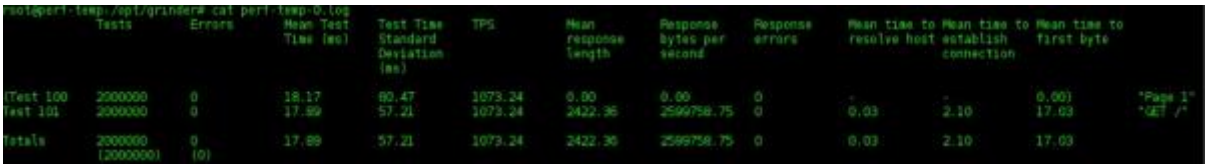

Рисунок 3.4 – Тестування навантаження за допомогою Grinder

2 Gatling Project - це інструмент для тестування продуктивності з відкритим вихідним кодом, який в основному розроблений та підтримується Stephane Landelle;

Рисунок 3.5 – Приклад звіту тестування навантаження с Gatling

3 Tsung – інструмент для тестування продуктивності з відкритим вихідним кодом. Tsung використовує Erlang. Tsung не надає графічного інтерфейсу для розробки або виконання тестів.

Основний виклик сценарію tsung робить такий висновок, представлений на рисунку 3.6. На рисунку 3.7 наведено статистичний звіт навантажувального тестування, а на рисунку 3.8 – графічний звіт;

```
blazemeter@perf-temp:~$ tsung
Usage: tsung <options> start|stop|debug|status
Options:
    -f <file>     set configuration file (default is ~/.tsung/tsung.xml)
                   (use - for standard input)
    -l <logdir>   set log directory where YYYYMMDD-HHMM dirs are created (default is ~/.tsung/log/)
    -i <id>       set controller id (default is empty)
    -r <command>  set remote connector (default is ssh)
    -s            enable erlang smp on client nodes
    -p <max>      set maximum erlang processes per vm (default is 250000)
    -m <file>     write monitoring output on this file (default is tsung.log)
                   (use - for standard output)
    -F            use long names (FQDN) for erlang nodes
    -w            warmup delay (default is 1 sec)
    -v            print version information and exit
    -6            use IPv6 for Tsung internal communications
    -x <tags>     list of requests tag to be excluded from the run (separated by comma)
    -h            display this help and exit
```

Рисунок 3.6 - Виклик сценарію tsung

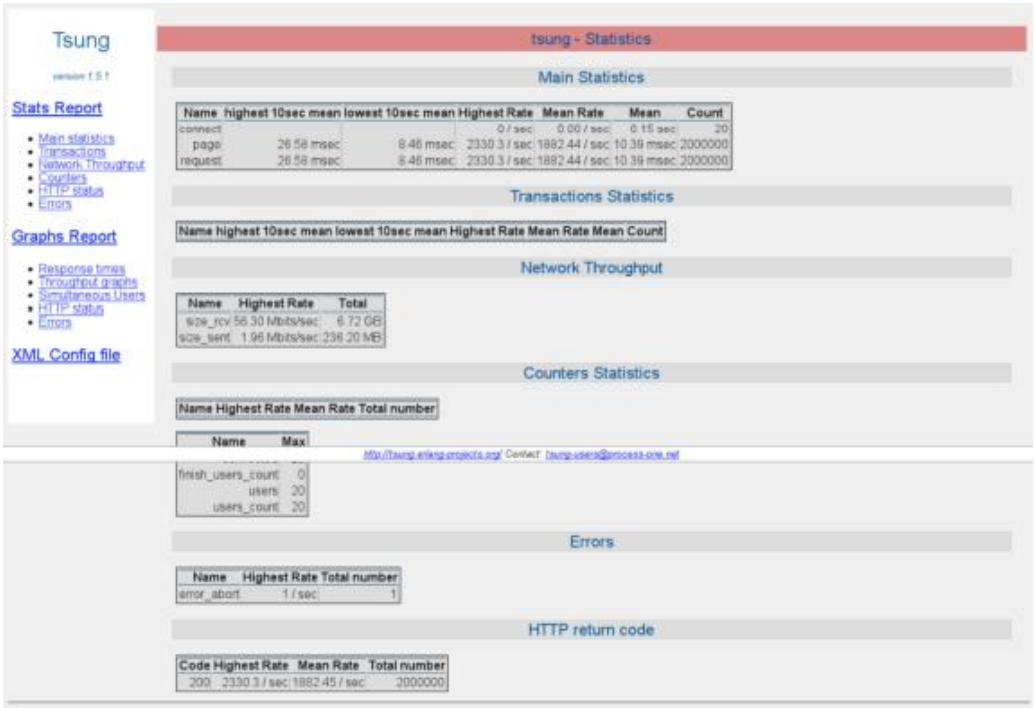

Рисунок 3.7 – Статичний звіт тестування навантаження

Рисунок 3.8 – Графічний звіт тестування навантаження

4 Apache JMeter™ - має зручний графічний інтерфейс, що значно полегшує розробку тестів та налагодження. Програма JMeter з агрегованим звітом за сценарієм навантаження (рисунок 3.9);

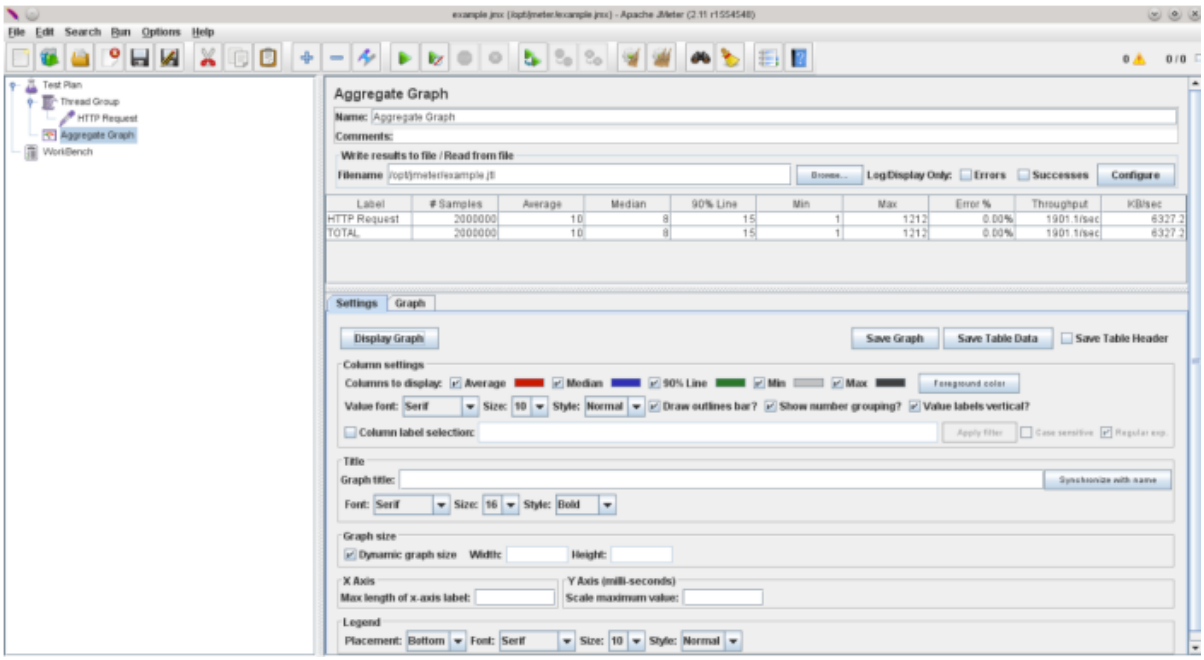

Рисунок 3.9 - Агрегований звіт сценарію навантаження JMeter

5 Locust [46] заснована на Python середовищі з відкритим вихідним кодом, яка дозволяє писати скрипти продуктивності чистою мовою Python.

Приклад базового тестового сценарію з висновком представлено рисунку 3.10.

```
from locust import HttpLocust, TaskSet, task
class SimpleLocustTest(TaskSet):
 @task
 def get_something(self):
 self.client.get("/")
class LocustTests(HttpLocust):
 task_set = SimpleLocustTest
```

```
→ JMeterVsLocust locust -f simple_locust_script.py --host=http://192.168.1.170:8080
[2017-11-18 14:46:38,458] Yuris-MBP-2.home/INFO/locust.main: Starting web monitor at *:8089
[2017-11-18 14:46:38,459] Yuris-MBP-2.home/INFO/locust.main: Starting Locust 0.8.1
```

Рисунок 3.10 – Приклад тестового сценарію Locust

Після виконання скрипта відображається докладна звітність (рисунок 3.11):

Рисунок 3.11 – Звітність тестового сценарію Locust

Порівняємо результати тесту навантаження цих інструментів з наступними показниками:

1 середній час відгуку (мс);

2 середня пропускна спроможність (запитів/сек);

3 загальний час виконання тесту (хвилин).

На рисунках 3.12, 3.13 представлені діаграми, що відображають середню відповідь та загальний час виконання тесту.

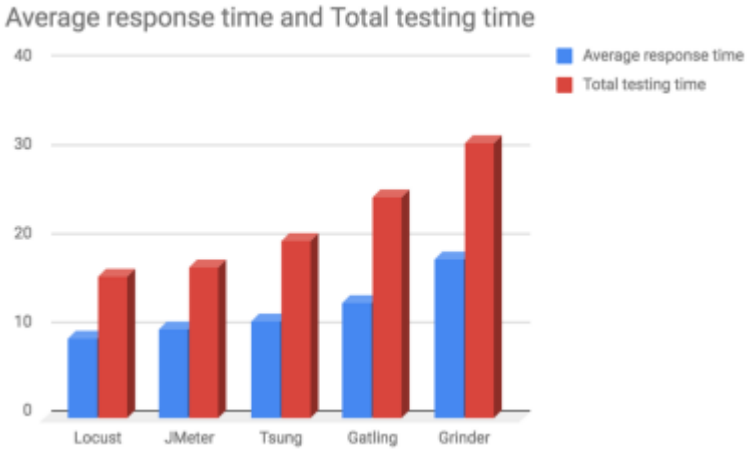

Рисунок 3.12 - Процесс тестування навантаження з використанням моделей

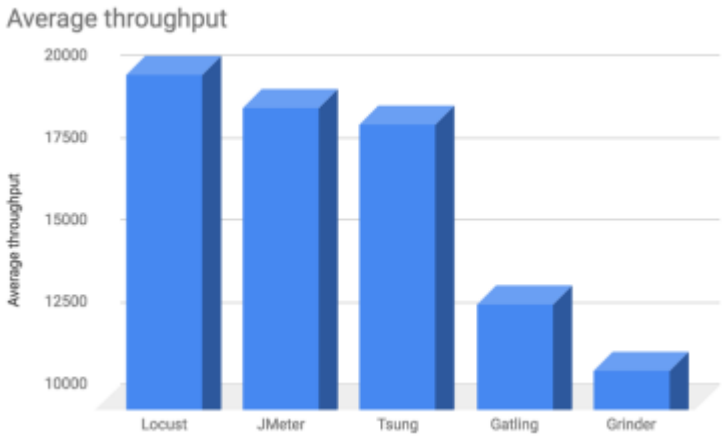

Рисунок 3.13 - Процесс тестування навантаження з використанням моделей

Як показано на графіках, у Locust найшвидший час відгуку з найвищим середнім значенням, за яким йдуть JMeter, Tsung і Gatling. У Grinder найповільніший час із найнижчою середньою пропускною здатністю.

У таблиці 3.2 наведено порівняння ключових функцій, пропонованих кожним інструментом тестування:

Таблиця 3.2 - Порівняльний аналіз інструментальних засобів навантажувального тестування

Особливість	The Grinder	Gatling	Tsung	JMeter	Locust
ОС	будь-яка	будь-яка	Linux/Unix	будь-яка	будь-яка
Графічний інтерфейс	консоль	рекордер	немає	повний	немає
Тест рекордер	TCP(HTTP)	HTTP	HTTP, Postgres	HTTP	немає
Тест мова	Python, Clojure	Scala	XML	XML	Python
Мова розришення	Python, Clojure	Scala	Erlang	Java, JS, Beanshell	Python

Продовження таблиці 3.2

Завантажити звіт	Console	HTML	HTML	CSV, XML, графіка та плагіни	HTML
Протоколи	HTTP, SOAP, JDBC, POP3, LDAP, SMTP	HTTP, JDBC, JMS	HTTP, WebDAV, MySQL, XMPP, WebSocket, LDAP	HTTP, FTP, JDBC, SOAP LDAP, TCP, SMTP, POP3	HTTP
Моніторинг хосту	ні	ні	так	так, з плагіном PerfMon	ні
Обмеження	Розуміння Python для розробки та редагування тестів.	Обмежена підтримка протоколів. Розуміння мови DSL. Не масштабована.	Підтримується тільки у системах Linux.	Звіти складно інтерпретувати.	Розуміння Python для розробки та редагування тестів.

Застосувавши вибрані інструментальні засоби для проведення тестування навантаження, були отримані наступні результати: час вибірки постійно знаходиться між 128 і 164 мс. Географічна відстань є найважливішим фактором, що впливає на час вибірки, якщо ваш сервер має достатні ресурси. У тестовому випадку сервер добре відреагував на 50 користувачів, які запитують протягом 10 секунд.

Тепер настав час спробувати збільшити навантаження та подивитися, наскільки добре система реагує.

На наступному етапі навантаження збільшується до 80 за 10 секунд. Нижче наведено результати (рисунок 3.14).

Label	Sample Time(ms)	Status	Bytes	Latency
HTTP Request	953		55065	546
HTTP Request	880		55065	479
HTTP Request	941		55065	529
HTTP Request	924		55065	502
HTTP Request	815		55065	398
HTTP Request	964		55065	567
HTTP Request	885		55065	489

Рисунок 3.14 – Результати тестування зі збільшенням навантаження

Сервер явно починає обтяжуватись запитами (рисунок 3.15). Запустимо тест JMeter для оцінки використання ресурсів VPS [47] (рисунок 3.16).

```
top - 16:52:25 up 5 days, 23:18,  1 user,  load average: 0.06, 0.16, 0.13
Tasks:  74 total,   1 running,  73 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.3 sy,  0.0 ni, 99.7 id,  0.0 wa   0 0 hi   0 0 si   0 0 st
KiB Mem:    501868 total,   409232 used,    92636 free,    28132 buffers
```

Рисунок 3.15 – Результати тестування зі збільшенням навантаження без використання JMeter

```
top - 16:45:57 up 5 days, 23:11,  1 user,  load average: 0.80, 0.35, 0.16
Tasks:  74 total,   3 running,  71 sleeping,   0 stopped,   0 zombie
%Cpu(s): 94.7 us,  4.7 sy,  0.0 ni,  0.3 id,  0.0 wa,  0.3 hi,  0.0 si,  0.0 st
KiB Mem:    501868 total,   410120 used,    91748 free,    28072 buffers
KiB Swap:        0 total,        0 used,        0 free.   240612 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
19233 www-data  20   0  270488  19892  12780 R 32.2  4.0   0:03.49 php5-fpm
19232 www-data  20   0  269720  19624  12772 S 31.9  3.9   0:03.83 php5-fpm
19231 www-data  20   0  271072  20664  12772 R 31.2  4.1   0:06.90 php5-fpm
```

Рисунок 3.16 – Результати тестування зі збільшенням навантаження з використанням JMeter

Очевидно, що є суттєве споживання ресурсів процесора та оперативної пам'яті. Для задоволення зростаючих робочих навантажень сервери повинні бути оптимізовані, або потрібно збільшувати ЦП. Також можна розмістити базу даних на іншому сервері.

Таким чином, можна підтримувати навантаження доти, доки не відбудеться істотне зниження продуктивності.

3.3 Тестування продуктивності та надійності бази даних КІС

Розглянемо тестування надійності та продуктивності КІС. Характеристики та список обладнання для проведення навантажувального тестування представлений у таблиці 3.3.

Таблиця 3.3 – Технічне забезпечення КІС

Позиція	Позначення	Найменування та технічні характеристики
1	Маршрутизатор	Asus RT-AC69Um
2	Сервер	Xeon E5-1635, 12 x 2.7 Гц, 2 процесорів, ОП – 8 ГБ
3	Зовнішнє сховище даних	BLOB в SharePoint Foundation
4	Джерело безперебійного живлення	Smart-UPs RT 3000 RM 230V
5	Робочі станції	Dell Precision 7510 4K IGZO (Precision 7k), Процесор: Intel Xeon E3-1635Mv5 2.9 GHz, Графічний адаптер: NVIDIA Quadro M2000M, Ядро: 1038 - 1197 MHz, nVIDIA ForceWare, ОП: 32768 Мбайт , DDR4,
6	Роутер	TP-LINK TL-WA901ND

Сервер додатків та БД для КІС повинен бути підключений до локальної мережі та мати наступні характеристики (не нижче): Intel Xeon 3,2Mhz; 8Гб ОЗУ; 500Гб HDD; 512Мб.

Загальними можливостями СУБД, що використовується під час роботи КІС, є:

- підтримка реляційної або об'єктно-реляційної моделі бази даних;
- підтримка міжнародного стандарту ANSI SQL-92 та вище;
- наявність засобів створення індексів та кластерів даних;
- автоматичне відновлення бази даних;
- підтримка мережевих протоколів TCP/IP;
- можливість контролю доступу до даних;
- централізоване керування обліковими записами користувачів;
- оптимізація запитів.

Тестування здійснюється ітераційно, збільшуючи на кожній ітерації кількість сесій тестів, що одночасно працюють. Тестування припиняється за одним із перерахованих вище критеріїв. На кожній ітерації фіксуються показники продуктивності. Після закінчення тестування за знятими показниками робляться висновки про кількість користувачів, що одночасно працюють.

Для результативного проведення процедури тестування продуктивності КІС необхідне певне технічне завдання та програмне забезпечення. Проводити збір та оцінку характеристик функціонування СУБД не потрібно.

Визначено види застосовуваного тестування продуктивності:

1 Визначення максимальної продуктивності – реалізується сценарієм № 2. Тест завершується, коли часи відгуку перевищили допустимі межі; кількість неуспішних операцій збільшилася до критичної (більше 10%); вичерпано системні чи апаратні ресурси. Результатом тестування є максимальний досягнутий рівень навантаження (позначається Lmax).

2 Тест надійності – реалізується сценарієм № 1. Критерієм успішності тестування є: відсутність деградації продуктивності системи під час тесту

(інтенсивності та часів відгуку); відсутність витоку ресурсів протягом тесту.

З Тест стійкості до відмови виконується на рівні типового навантаження, який зазвичай встановлюється на рівні 70% від максимальної Lmax.

Критеріями успішного проходження системою тесту є [6]:

- система зберегла доступність у функціоналі, не пов'язаному з функціями суміжної системи;

- система відновилася протягом необхідного часу відновлення доступності.

Критеріями успішного завершення тестування навантаження є виконання всіх запланованих тестів і отримання даних моніторингу.

Виконання процедури обліку кредитування клієнта в системі за одночасної участі 10 користувачів. Алгоритм процедури представлений стандартним чином (рисунок 3.17).

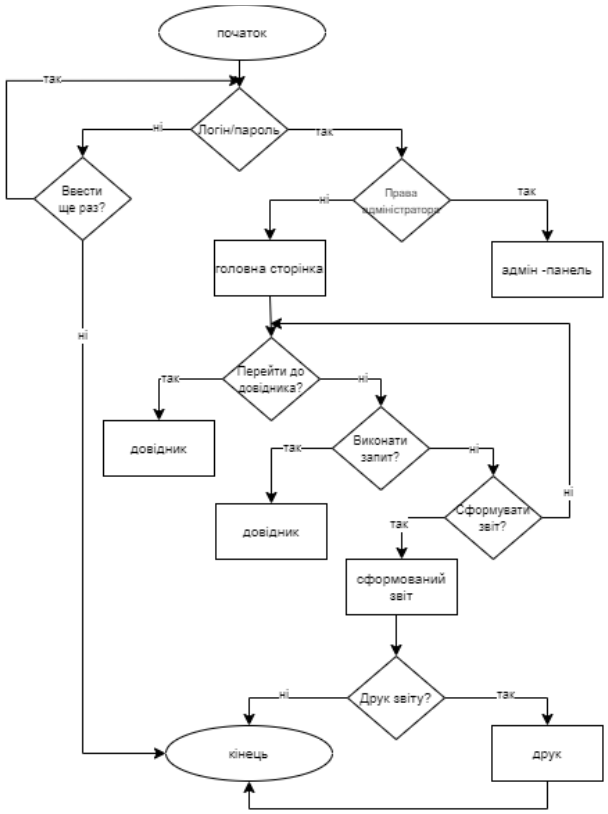

Рисунок 3.17 – Алгоритм процедури тестування за сценарієм 1

Імітація DDoS атаки [13] (рисунок 3.18). Одночасна робота в одному із довідників системи максимальної кількості користувачів – 30-50 од.

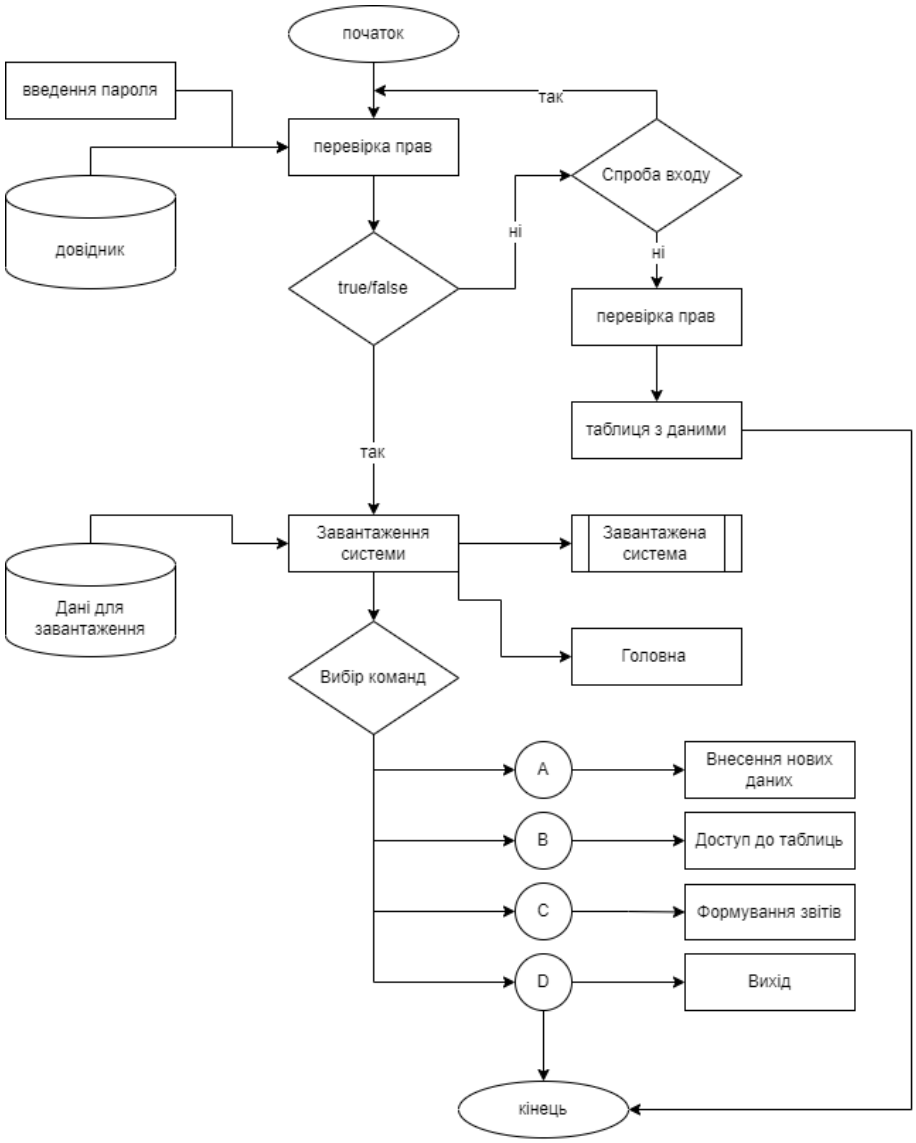

Рисунок 3.18 – Алгоритм роботи користувача під час імітації DDoS – атаки

За умови успішності сценарію кількість користувачів збільшується 10 од. та тестування триває. Визначається межа можливостей КІС.

Перед початком ітераційного навантажувального тестування повинна бути розроблена програма та методика випробувань, що включає:

- опис цілей та завдань тестування;
- опис об'єкта тестування, включаючи опис основних бізнес-процесів

системи;

- архітектурну схему об'єкту тестування;
- короткий опис підходу до тестування та способів генерації навантаження;
- опис профілів навантаження, включаючи перелік операцій, виконуваних віртуальними користувачами та зовнішніми системами через відповідні інтерфейси; інтенсивність операцій, що виконуються віртуальними користувачами та зовнішніми системами [5];
- список запланованих тестів;
- типові сценарії тестування;
- тестові скрипти;
- вимоги до тестових даних;
- вимоги до продуктивності;
- вимоги до моніторингу продуктивності;
- метрики продуктивності;
- обмеження тестування;
- ключові показники досягнення цілей;
- ризики проекту.

Навантажувальне тестування КІС виконується на заздалегідь підготовленій базі з наступними параметрами:

1 кількість співробітників, що одночасно працюють, до 50;
2 кількість об'єктів обслуговування (юридичних та фізичних осіб) – 10 000;
3 кількість видів обслуговування кожному за об'єкта – 4;
4 кількість контрольованих показників кожного об'єкта 4-5;
5 кількість показників напрацювання для кожного об'єкта – 4-5.

Після задоволення вимог щодо технічного та програмного забезпечення запускається навантажувальний тест, та оцінюється швидкість виконання операцій та інші параметри відповідно до запропонованої методики. Далі результат порівнюється з "еталонними" значеннями. За результатами

порівняння приймається рішення про необхідність оптимізації та випуск оновлень.

Передача інформації в КІС здійснюється шляхом електронної пошти, локальною мережею або у формі документів. Достовірність введення даних буде реалізована за допомогою лічильного контролю.

Автентифікація користувачів у КІС базується на механізмах, вбудованих у використовувану СУБД (Microsoft SQL Server або Sybase Adaptive Server). При цьому використовується концепція «наскрізної» аутентифікації та авторизації, яка передбачає використання єдиного облікового запису як на рівні системи, так і на рівні СУБД.

Для захисту інформації від несанкціонованого доступу та зміни особами, які мають адміністративні повноваження, в системі вжито спеціальних заходів щодо поділу сфер відповідальності адміністраторів та обмеження їх повноважень.

Для визначення максимально можливого навантаження на систему необхідно отримати зразкову оцінку реально працюючих користувачів із конфігурацією. Зробити це можна за такою методикою. Припустимо, що користувача для роботи з одним документом йде 1 хвилину. Тоді загальну кількість реально працюючих користувачів можна одержати із співвідношенням часу введення документа до загального часу роботи з документом. При цьому під загальним часом роботи з документом розуміється сумарний час виконання підготовчих запитів до бази заповнення реквізитів документа, часу введення документа в базу, а також бездіяльності, що дорівнює 1 хвилині. Нижче наведемо формулу:

$$Rel = \frac{t_{save}}{t_{prep}+t_{pause}+t_{save}} \quad (3.8)$$

де $tprep$ – сумарний час необхідної послідовності запитів до бази, що імітує заповнення реквізитів документа;

- $tpause$ – пауза на хвилину;

- $tsave$ – час збереження документа.

Після проведення тестування необхідно оцінити відношення часу $tsave$ до загального часу роботи користувача.

Загальна кількість користувачів:

$$P = s/Rel, \tag{3.9}$$

де s – максимально досягнута кількість сесій, що одночасно працюють.

Розглянемо побудову марківської моделі для тестування навантаження КІС при роботі з базами даних.

Нехай існує деяка впорядкована безліч запитів Q до деякої бази даних. Воно включає всі запити qi (i = 1...n, n - кількість запитів), виконані в БД, а також спеціальний «порожній запит» q, який означає, що в даний момент часу запитів до БД не надходило, і модель переходить початковий стан (стан очікування запиту). Кожному запиту БД зіставляється стан цього запиту si, а початковий стан моделі позначимо s0. Далі з файлу журналу аналізується заданий часовий інтервал T, розбитий з певним кроком tj, значення tj (j = 1…m, m – число значень на зазначеному інтервалі).

Таким чином, представлений файл журналу можна формалізувати у вигляді безлічі трійок виду:

$$L = \{l_j | l_j = (t_j, id, q_i)\}, \tag{3.10}$$

Далі розглянемо процес побудови марківської моделі по множині L. На початку процесу аналізу модель перебуває у стані s0. Якщо у момент часу tj прийшов запит qi від клієнта id, то одна трійка із зазначеними параметрами відзначається в множині L, і далі не розглядається. Модель перетворюється на стан si, у своїй можливість переходу у зазначений стан буде:

$$P(s_i) = \frac{n_{si}}{m}, \quad (3.11)$$

де nsi - кількість переходів у стан si;

m - загальна кількість розбиття часового інтервалу T.

В результаті обчислень формується четвірка виду:

$$(s_{n-1}, s_n, q_i, p), \quad (3.12)$$

де $sn-1$ – попередній стан моделі;

sn - новий стан моделі;

qi - запит, що викликав перехід у стан sn;

p - розрахункове значення ймовірності.

Безліч трійок є марковською моделлю навантажувального тестування КІС при роботі з базами даних. Далі зазначений процес повторюється, використовуючи як базовий стан sn.

Для реалізації алгоритму введемо операції Mark(li) – позначки певного елемента множини, операцію M(L), що повертає як результат потужність множини L, імовірність(qi) - результатом операції є ймовірність, розрахована за формулою (3.11). Тоді схема алгоритму побудови цієї моделі виглядає, як показано на рисунок 3.19. Розглянемо далі безпосередньо процес тестування навантаження СУБД з використанням запитів КІС. Нехай існує марківська модель (рисунок 3.20).

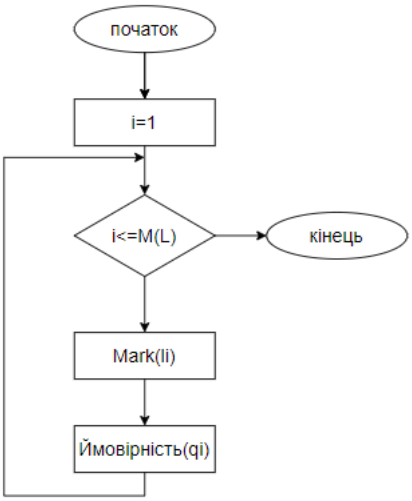

Рисунок 3.19 – Схема алгоритму побудови марківської моделі

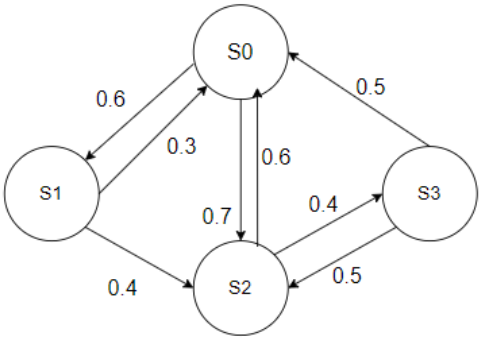

Рисунок 3.20 – Марківська модель (граф [50])

За підсумками проходження тесту будується відношення час відповіді на конкретний запит. Це ставлення складається з безлічі пар, перша компонента яких запит, друга – час відгуку СУБД [48] даний запит.

Це при постійних параметрах налаштування і структури таблиць і запитів має властивість функціональності, тобто. кожному запиту відповідає свій заданий час відповіді. При проведенні тестів важливим питанням є отримання пропорційних результатів відповіді на аналогічних тестах продуктивності. Актуальність цього питання пов'язана з тим, що результати тесту залежать не тільки від параметрів апаратного та програмного налаштування сервера СУБД, а й від побудови самого тесту.

Налаштування СУБД доцільно виконувати за результатами двох або більше продуктивних тестів. Тому, крім тестування, заснованого на марківських моделях поведінки клієнтів СУБД, доцільно проводити тестування одним або декількома синтетичними тестами, структура запитів яких найбільше збігаються зі структурою запитів КІС. Для з'ясування ступеня цієї відповідності, побудовані раніше марківські моделі, необхідно доповнити інформацією структуру запитів КІС.

Тестування продуктивності дозволяє визначити ступінь готовності системи до позаштатних ситуацій (відмова обладнання, DDoS - атаки), рівень її надійності та здатність до самовідновлення. Також навантажувальні випробування допомагають розробити комплекс адекватних заходів для підвищення продуктивності системи, її стійкості та захищеності корпоративного оточення.

3.4 Сценарій тестування навантаження та формування критеріїв на доопрацювання

Як приклад зробимо навантаження тестування КІС під 30 користувачами одночасно, які послідовно входять в систему з інтервалом 3 секунди. Динаміка подачі навантаження представлена рисунку 3.21.

Рисунок 3.21 – Динаміка навантаження

Усі 30 користувачів починають працювати з системою через 90 секунд з моменту початку тесту навантаження. Розподіл часу відгуку по транзакціях щодо початку тесту навантаження наведено на рисунку 3.22.

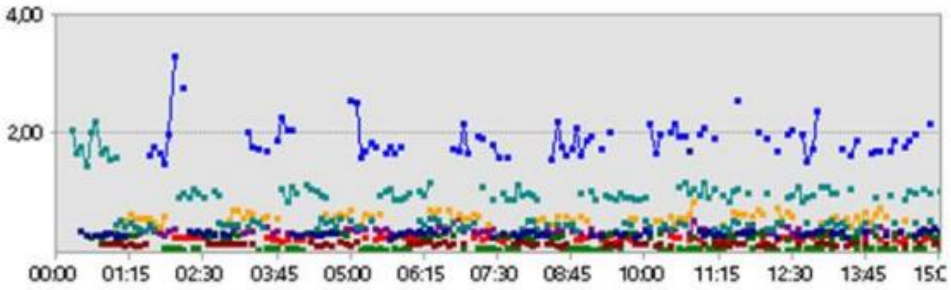

Рисунок 3.22 – Час відгуку системи

У процесі виконання сценарію усі транзакції було виконано успішно. У таблиці 3.4 наведено дані щодо розподілу часу відгуку.

Таблиця 3.4. Статистика виконання сценарію

Транзакція	Min (сек)	Aver (сек)	Max (сек)
Ввести некоректні дані логіну/паролю та перевірити аут	0.012	0.014	0.017
Увійти в систему під паролем	0.70	3.06	45.7
Відкрити розділ «Введення даних»	0.11	2.89	32.1
Відкрити «Паспорт»	0.12	0.16	4.11
Перейти у таблицю 1	0.054	0.37	21.6
Ввести 50 значень та зберегти	0.11	1.55	42.4
Перейти на рівень ОМ	0.44	1.11	33.6
Перейти у таблицю 4	0.12	0.55	8.4
Ввести 50 значень та зберегти	0.12	0.43	1.3
Перейти у розділ «Друковані форми»	0.04	0.08	0.3
Обрати додаток та версію	0.43	1.33	3.56

Продовження таблиці 3.4

Обрати таблицю 1 та сформувати	3.44	8.7	54.66
Обрати таблицю 4 та сформувати	1.3	2.2	2.77
Обрати таблицю 2 та сформувати	2.4	8.84	34.2
Вийти з системи	0.002	0.01	0.04
Увійти в систему з правами адміна	0.032	3.76	28.1
Відкрити вкладку «Моніторинг»	0.14	1.45	22.6
Відкрити розділ «Звіт – виконано»	0.44	2.87	32.1
Обрати 3 квартал 2021 року	0.65	4.63	20.7
Сформувати пояснювальну записку	0.1	0.16	0.21
Відкрити розділ «Звіт»	1.03	2.0	3.55
Вийти з системи	0.023	0.033	0.70

За результатами таблиці 3.4 видно, що час відгуку системи відповідає вимогам продуктивності при навантаженні 30 користувачів.

Найменш продуктивними є транзакції: «Вибрати таблицю 1 та Сформувати», «Вибрати таблицю 2 та Сформувати», «Відкрити вкладку «Моніторинг», рекомендується звернути увагу на їхню оптимізацію.

У процесі тестування навантаження необхідно знімати лічильники продуктивності з сервера додатків. Результати подано на рисунку 3.23 та рисунку 3.24.

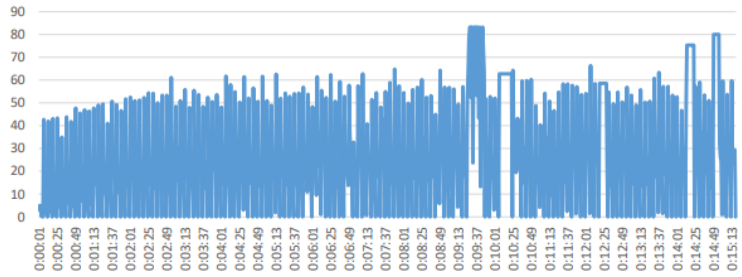

Рисунок 3.23 – Споживання ресурсів CPU сервера додатків

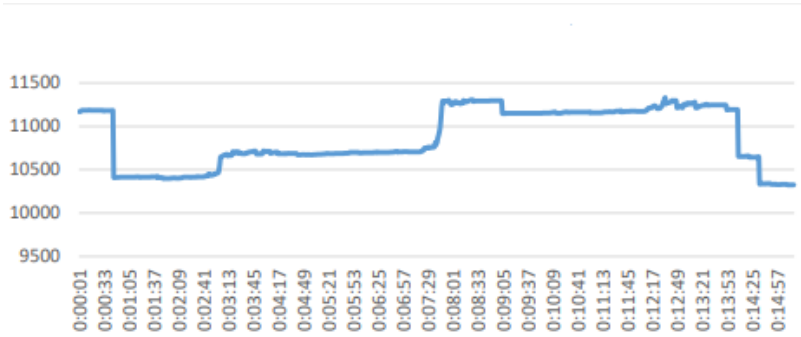

Рисунок 3.24 – Споживання пам'яті сервера програм

Також необхідно знімати лічильники продуктивності із сервера бази даних (БД). Результати подано на рисунку 3.25 та рисунку 3.26.

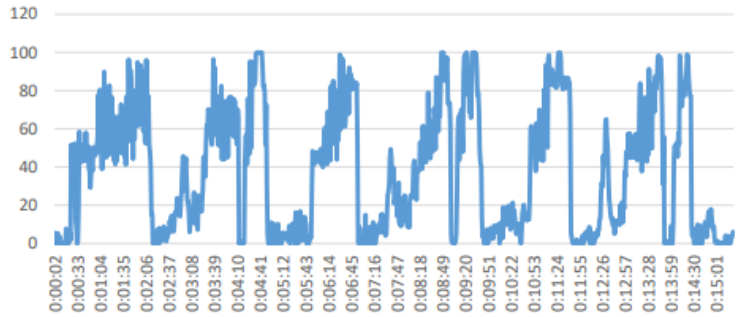

Рисунок 3.25 – Споживання ресурсів CPU серверу бд

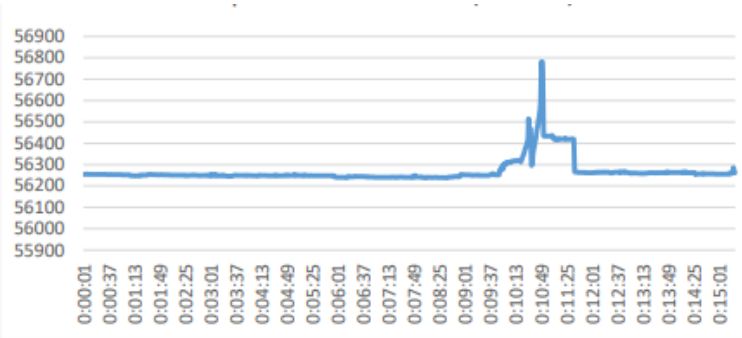

Рисунок 3.26 – Споживання пам’яті сервера бд

З графіків видно, що протягом усього виконання сценарію спостерігається середнє завантаження CPU сервера в додатку 60% з піками до

85%, середнє сервера БД 60% з піками до 100%. Споживання пам'яті сервером БД та додатків знаходиться на середньому рівні 10.4-11.3 і 56.2-56.8 гб.

В результаті проведеного тесту навантаження можна зробити наступні висновки:

- CPU сервера БД [49], ймовірно, не є вузьким місцем, що обмежує продуктивність системи;

- CPU сервера додатків, ймовірно, є вузьким місцем, що обмежує продуктивність системи;

- споживання пам'яті сервера додатків та сервера БД не є вузьким місцем, що обмежує продуктивність системи;

- на основі аналізу лічильників продуктивності робота дискової підсистеми не є вузьким місцем; аналогічно завантаження клієнтської станції.

- як рекомендацію розробникам системи слід звернути увагу на оптимізацію найменш продуктивних транзакцій «Вибрати таблицю 1 та Сформувати», «Вибрати таблицю 2 та Сформувати», Відкрити вкладку «Моніторинг».

-

3.5 Інтеграційне тестування корпоративної інформаційної системи

Проведено інтеграційне тестування за допомогою платформи Citrus Framework.

Citrus - це інтегроване середовище тестування, написане на Java, яке тестує КІС щодо відповідності середовищу клієнта. Інструмент імітує навколишні системи через різні порти (HTTP, JMS, TCP/IP, SOAP тощо), щоб виконати автоматичне наскрізне тестування варіанта використання.

Складаючи можливі сценарії, здійснюється перевірка КІС, щоб мати можливість відправляти повідомлення за різними протоколами та виконувати всю кореляцію повідомлень, які надходять і зрештою прибувають до цільового пункту призначення.

Щоб змоделювати один із цих сценаріїв, був використаний Citrus

Framework. Для перевірки його роботи була змодельована структура каталогів (рисунок 3.27):

- src/citrus/java - місце розташування згенерованих тестових прикладів Testng;
- src /citrus /resoruces - розташування всієї конфігурації, необхідної для фреймворку;
- src/citrus/tests- розташування всіх певних тестових випадків;
- lib – розташування всіх бібліотек залежностей;
- log – розташування, в якому Citrus Framework [51] зберігатиме логи (фрагменти корисного навантаження);
- build.xml - скрипт складання ant.

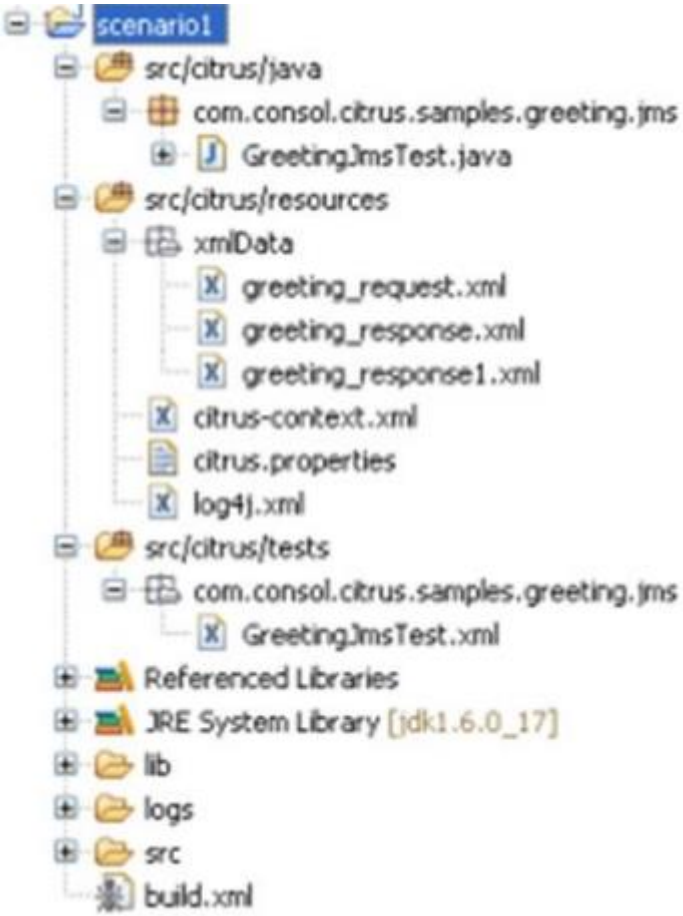

Рисунок 3.27 – Структура каталогів

На рисунку 3.28 наведено приклад використовуваного build.xml.

```
<project name="greetings" basedir="." default="citrus.run.tests">

    <property file="src/citrus/resources/citrus.properties"/>

    <path id="citrus-classpath">
        <pathelement path="src/citrus/java"/>
        <fileset dir="lib">
            <include name="*.jar"/>
        </fileset>
    </path>

    <typedef resource="citrustasks" classpath="lib/citrus-ant-tasks-1.1-SNAPSHOT.jar"/>

    <target name="compile.tests">
        <javac srcdir="src/citrus/java" classpathref="citrus-classpath"/>
        <javac srcdir="src/citrus/tests" classpathref="citrus-classpath"/>
    </target>

    <target name="citrus.run.tests" depends="compile.tests" description="Runs all Citrus
        <citrus suitename="${testsuite.name}" package="com.consol.citrus.samples.*"/>
    </target>

    <target name="citrus.run.single.test" depends="compile.tests" description="Runs a si
        <touch file="test.history"/>

        <loadproperties srcfile="test.history"/>

        <echo message="Last test executed: ${last.test.executed}"/>

        <input message="Enter test name:" addproperty="testclass" defaultvalue="${last.t

        <propertyfile file="test.history">
            <entry key="last.test.executed" type="string" value="${testclass}"/>
        </propertyfile>

        <citrus suitename="citrus-samples" test="${testclass}"/>
    </target>

    <target name="create.test" description="Creates a new empty test case">
        <input message="Enter test name:" addproperty="test.name"/>
        <input message="Enter test description:" addproperty="test.description" defaultv
        <input message="Enter author's name:" addproperty="test.author" defaultvalue="${
        <input message="Enter package:" addproperty="test.package" defaultvalue="${defau
        <input message="Enter framework:" addproperty="test.framework" defaultvalue="tes

        <java classname="com.consol.citrus.util.TestCaseCreator">
            <classpath refid="citrus-classpath"/>
            <arg line="-name ${test.name} -author ${test.author} -description ${test.des
        </java>
    </target>

    <target name="create.html.doc" description="Creates test documentation in html">
        <mkdir dir="target"/>

        <java classname="com.consol.citrus.doc.HtmlTestDocGenerator">
            <classpath refid="citrus-classpath" />

            <arg value="src/citrus/tests"/>
            <arg value="target/CitrusTests.html"/>
            <arg value="Citrus Test Documentation"/>
            <arg value="logo.png"/>
            <arg value="Overview"/>
        </java>

        <copy todir="target" file="src/citrus/resources/logo.png"/>

        <zip destfile="target/CitrusTests.zip">
            <fileset dir="target">
                <include name="CitrusTests.html"/>
                <include name="logo.png"/>
            </fileset>
        </zip>
    </target>

    <target name="create.xls.doc" description="Creates test documentation in excel">
        <mkdir dir="target"/>

        <java classname="com.consol.citrus.doc.ExcelTestDocGenerator">
            <classpath refid="citrus-classpath" />

            <arg value="src/citrus/tests"/>
            <arg value="CitrusTests"/>
            <arg value="Citrus Test Documentation"/>
            <arg value="Citrus TestFramework"/>
            <arg value="Consol* Software GmbH"/>
        </java>
    </target>
</project>
```

Рисунок 3.28 – Приклад використовуваного build.xml

ПО КІС за сценарієм отримує нове повідомлення у черзі. Якийсь процес отримує повідомлення, виконує його перетворення/перевірку та передає його. Зрештою, нове повідомлення буде надіслано в іншу чергу.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<spring:beans xmlns="http://www.citrusframework.org/schema/testcase" xmlns:spring="http:.
    <testcase name="GreetingJmsTest">
        <meta-info>
            <author>Eric Elzinga</author>
            <creationdate>2010-03-25</creationdate>
            <status>FINAL</status>
            <last-updated-by>Eric Elzinga</last-updated-by>
            <last-updated-on>2010-03-28T00:00:00</last-updated-on>
        </meta-info>

        <variables>
            <variable name="correlationId" value="citrus:randomNumber(10)"></variable>
            <variable name="user" value="Eric"></variable>
        </variables>

        <actions>
            <send with="sendGreeting">
            <message>
                <resource file="classpath:xmlData/greeting_request.xml" />
            </message>
            <header>
                <element name="Operation" value="sayHello"/>
                <element name="CorrelationId" value="${correlationId}"/>
            </header>
        </send>

        <receive with="receiveGreeting">
        <selector>
            <value>CorrelationId = '${correlationId}'</value>
        </selector>
        <message>
          <resource file="classpath:xmlData/greeting_response.xml" />
          <ignore path="//tns:GreetingRequestMessage/tns:Text" />
        </message>
        <header>
            <element name="Operation" value="sayHello"/>
            <element name="CorrelationId" value="${correlationId}"/>
        </header>
    </receive>
</actions>
</testcase>
</spring:beans>
```

Рисунок 3.29 – Файл конфігурації

Щоб увімкнути корисне навантаження тесту, можна використовувати наступні команди: resource file, <tns: GreetingRequestMessage тощо.

Таким чином, була проведена перевірка повідомлень XML та отримуємо результат, наведений на рисунку 3.30:

```
[citrus] 4547   INFO  port.LoggingReporter| TEST FINISHED: GreetingJmsTest
[citrus] 4547   INFO  port.LoggingReporter| ----------------------------------------
[citrus] 4563   INFO  port.LoggingReporter| FINISH TESTSUITE citrus-samples-greeting
[citrus] 4563   INFO  port.LoggingReporter| ----------------------------------------
[citrus] 4563   INFO  port.LoggingReporter| ________________________________________
[citrus] 4563   INFO  port.LoggingReporter|
[citrus] 4563   INFO  port.LoggingReporter| CITRUS TEST RESULTS
[citrus] 4563   INFO  port.LoggingReporter|
[citrus] 4563   INFO  port.LoggingReporter|  GreetingJmsTest ..........................
[citrus] 4563   INFO  port.LoggingReporter|
[citrus] 4563   INFO  port.LoggingReporter| Total number of tests: 1
[citrus] 4563   INFO  port.LoggingReporter| Skipped: 0 (0.0%)
```

Рисунок 3.30 – Результат роботи тесту

Даним прикладом була продемонстрована можливість перевірки – чи можна помістити повідомлення у чергу та отримати відповідь назад у чергу;

перевіряє відповідь xml відповідність даному визначенню схеми (XSD).

На рисунку 3.31 наведено результати тестування інтеграції за допомогою soapUI:

Test Step	min	max	avg	last	cnt	tps	bytes	bps	err	rat
getAccountsByCustomer	508	6695	1 617,6	6695	78	0,25	68718	225	0	0
getBillingStatisticsByAccount	571	6939	1 513,46	653	78	0,25	69810	229	0	0
getCustomerInfoByID	527	14456	1 278,11	850	78	0,25	68562	225	0	0
getDiscountStatisticsByProduct	586	10342	1 384,21	786	78	0,25	69420	228	0	0
getProductInstanceByPhone	753	18080	2 047,69	943	78	0,25	365742	1201	0	0
getProductsByAccount	569	16439	1 699,6	762	78	0,25	69108	227	0	0
getProductsByAccountReduced	601	5686	1 110,41	794	78	0,25	69654	228	0	0
getUsageDetailsByProduct	629	7785	1 593,73	746	78	0,25	69420	228	0	0
getUsageStatisticsByProduct	494	16444	1 581,05	807	78	0,25	69654	228	0	0
isPostpaid	545	7742	1 391,91	747	78	0,25	60216	197	0	0
setUserInfo	517	5626	1 468,1	677	78	0,25	69498	228	0	0
validateCustomer	531	7661	1 563,52	4174	78	0,25	70356	231	0	0
TestCase:	6831	123895	18 249,42	18634	78	0,25	1120158	3680	0	0

Рисунок 3.31 – Результати тесту навантаження Wizard API [52]

Для тестування було визначено 1000 запитів з одночасним надсиланням 5 запитів на сервер. Середній час повного завантаження сторінки під час роботи одночасно 50 осіб становить близько 25 секунд, що у межах допустимості. Швидкість передачі між системами визначається інтервалах: до 30 сек – висока, 30 - 50 – середня, від 50 – низька. Таким чином, на даному прикладі була продемонстрована можливість приймального тестування показати, що всі модулі інтегровані між собою та мають гарну взаємодію, при роботі не виявлено дефекти. І це підтверджується досвідченим шляхом.

3.6 Аналітичні показники тестування безпеки корпоративних інформаційних систем

КІС є складною структурою, що об'єднує в собі різні послуги, необхідні для функціонування компанії. Ця структура змінюється: з'являються нові елементи, змінюється конфігурація існуючих. У міру зміни та збільшення КІС все більш важливим та актуальним завданням стають забезпечення інформаційної безпеки та захист критично важливих для компанії ресурсів.

З метою виявлення недоліків захисту різних компонентів та визначення потенційних атак на інформаційні ресурси проводиться аналіз захищеності. Найбільш ефективний спосіб аналізу – тестування на проникнення, у ході якого моделюється реальна атака зловмисників. Такий підхід дозволяє об'єктивно оцінити рівень захищеності корпоративної інфраструктури та зрозуміти, чи можуть протистояти атакам засоби захисту, що застосовуються в компанії.

Для аналізу захищеності КІС наведемо огляд найпоширеніших недоліків безпеки, практичні приклади їх експлуатації та опис ймовірних векторів атак, а також рекомендації щодо підвищення рівня захищеності.

Для підбиття статистики було сформовано підсумкову вибірку низки українських компаній із різних галузей економік, у своїй більшу частину склали промислові, фінансові та транспортні компанії (рисунок 3.32).

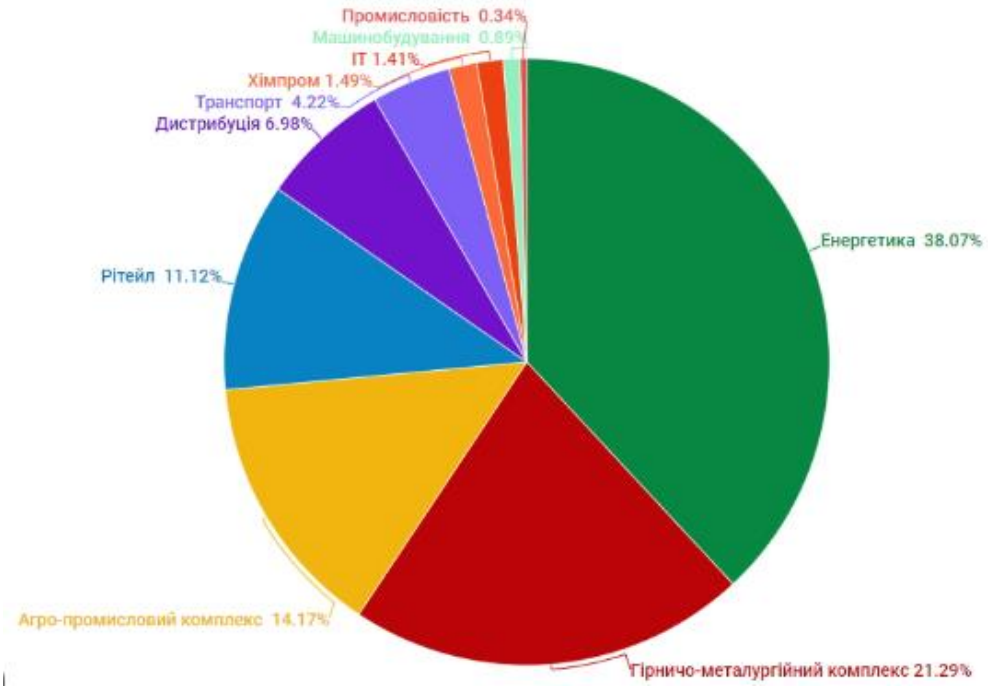

Рисунок 3.32 – Розподіл досліджених систем за галузями економіки

Аналіз захищеності КІС проводиться шляхом зовнішнього та внутрішнього тестування на проникнення, щоб сформувати коректну оцінку

рівня захищеності відтворюються умови, максимально наближені до умов реальної атаки. У ході зовнішнього тестування моделюються дії потенційного зловмисника, який не має привілеїв у системі і діє з інтернету. У цьому випадку перед експертами поставлено завдання подолати мережевий периметр і отримати доступ до ресурсів локальної мережі.

Внутрішнє тестування передбачає, що порушник діє із сегмента локальної мережі, яке має на меті контроль над інфраструктурою або над окремими критично важливими ресурсами, які визначає замовник. Комплексне тестування на проникнення передбачає обидва види робіт. Деякі компанії виконували аналіз захищеності бездротових мереж та оцінку обізнаності персоналу в питаннях інформаційної безпеки [53] (на 2022 рік).

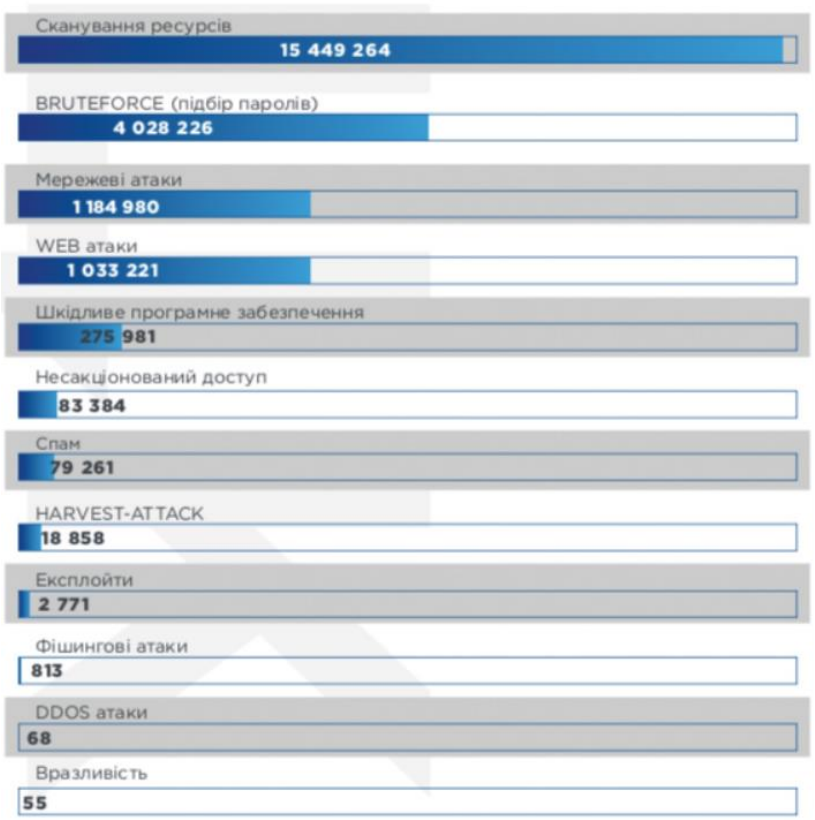

Рисунок 3.33 – Види кібер – атак за 2020 рік

При аналізі захищеності експерти виявляють різні вразливості та недоліки механізмів захисту, які можна розділити на чотири категорії:

- недоліки конфігурації;
- відсутність оновлень безпеки;
- уразливості у коді веб-додатків;
- недоліки парольної політики.

Кожній уразливості надається рівень ризику, який розраховується відповідно до системи класифікації CVSS 3.0. Практично у всіх системах було виявлено критично небезпечні вразливості, переважно пов'язані з вадами парольної політики.

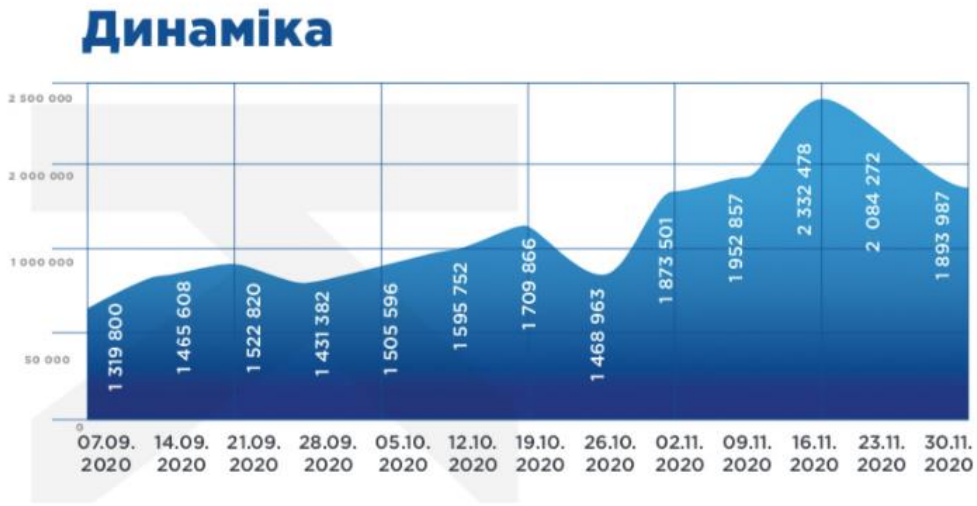

Рисунок 3.34 – Максимальний рівень небезпеки вразливостей

Важливо враховувати, що роботи з тестування на проникнення проводяться методом чорної скриньки, тому неможливо виявити уразливості, що у системі. В інфраструктурі кожної компанії могли бути недоліки захисту, зумовлені відсутністю своєчасного оновлення ПЗ, уразливістю в коді веб-додатків та використанням словникових паролів, які не були виявлені під час аналізу. Метою тестування на проникнення є пошук всіх без винятку недоліків системи, а отримання об'єктивної оцінки рівня її захищеності від атак порушників.

Найчастіше проникнути у внутрішню мережу можна кількома способами.

За статистикою в половині досліджуваних компаній є спосіб подолати мережевий периметр всього за один крок; як правило, він полягав у експлуатації вразливості у веб-додатку.

Суб`єкти кібербезпеки
(у межах компетенції)

Суб'єкт кібербезпеки	Кількість кіберінцидентів
Приватний сектор	237 441
Критична інфраструктура	63 505
Державні органи	2 938 475

Рисунок 3.35 – Суб’єкти кібербезпеки

Типовий сценарій атаки є підбір словникового облікового запису користувача КІС і подальша експлуатація вразливості, що виникла через помилки в коді веб-програми, наприклад можливості завантаження на сервер довільних файлів.

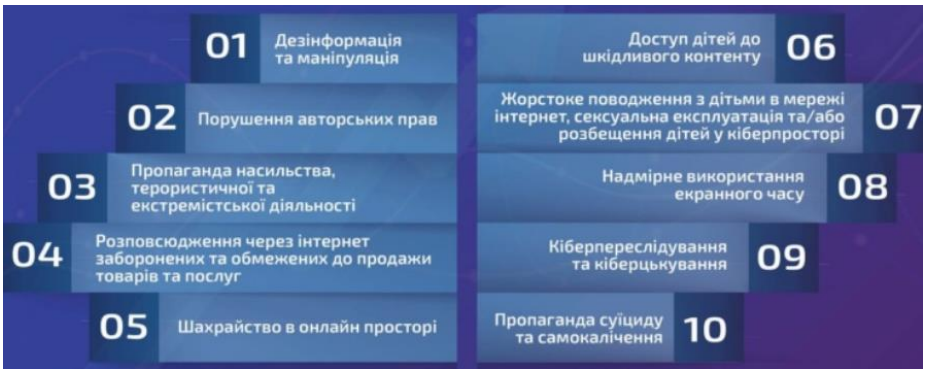

Рисунок 3.36 – Класифікація онлайн - загроз

Рекомендується регулярно проводити аналіз захищеності веб-застосунків. Чим складніше веб-додаток і чим більше у нього різних функцій, тим вища

ймовірність того, що розробники припустилися помилки, яка дозволить зловмисникові провести атаку. Частково такі помилки виявляються в рамках тестування на проникнення, але найбільше їх число може бути виявлено тільки при перевірці програми методом білої скриньки, що передбачає аналіз вихідного коду. Для виправлення вразливостей зазвичай потрібно внести зміни до коду, на що може знадобитися значний час. Щоб зберегти безперервність бізнес-процесів, рекомендується застосовувати міжмережевий екран рівня додатків (web application firewall), який не дозволить експлуатувати вразливість доти, доки її не усунули, а також захистить від нових і ще не знайдених вразливостей.

Інші вектори полягали переважно у підборі словникових паролів до різних систем - Outlook Web App (OWA), VPN-серверів та робочих станцій, а також у використанні недоліків конфігурації мережного обладнання. Подолання периметра потенційно можливе і через застарілі версії, які містять вразливості, що дозволяють отримати контроль над сервером. Для багатьох таких уразливостей є загальнодоступні експлойти, але їх експлуатація може порушити роботу систем, тому замовники зазвичай не погоджуються на проведення подібних перевірок у межах тестування на проникнення.

Як приклад експлуатації візьмемо словникові паролі користувачів. У ході тестування на проникнення виявлено, що доступу до сервісу OWA використовується доменний обліковий запис test:test1234.

Підключившись до OWA, було завантажено автономну адресну книгу (Offline Address Book), де містяться ідентифікатори користувачів домену. Підібравши словниковий пароль до облікового запису одного з користувачів, можна підключитися до шлюзу віддалених робочих столів (RDG) і за допомогою протоколу RDP отримати доступ до комп'ютера співробітника компанії та внутрішньої мережі.

Для повного представлення тестування вразливості КІС проаналізуємо інтерфейс керування обладнанням, доступний із зовнішніх мереж, словникові паролі користувачів.

Один із найпоширеніших варіантів проведення успішних атак у рамках тестування — виявлення на мережному периметрі інтерфейсів систем, які мають бути доступні виключно із внутрішньої мережі. Важливо правильно визначати межі мережного периметра та стежити за станом захищеності кожного компонента системи.

За підсумками статистики десятка найпоширеніших вразливостей на мережевому периметрі мало змінюється рік у рік. У низці компаній широко поширене використання відкритих протоколів передачі, зокрема доступу до інтерфейсів адміністрування. Зловмисник може перехопити облікові дані, що передаються за відкритими протоколами без використання шифрування та отримати доступ до відповідних ресурсів. Більш ніж у половині досліджуваних систем зовнішньому порушнику доступні інтерфейси віддаленого доступу, управління обладнанням та підключення до СУБД.

Як рекомендації можна запропонувати такі:

- обмежити кількість сервісів на мережевому периметрі, переконатися, що відкриті для підключення інтерфейси дійсно мають бути доступні всім інтернет-користувачам;

- регулярно проводити інвентаризацію ресурсів, доступних для підключення з Інтернету. Уразливості можуть з'явитися будь-якої миті, оскільки конфігурація інфраструктури постійно змінюється, у ній з'являються нові вузли, нові системи, і не виключені помилки адміністрування;

- відмовитися від використання простих та словникових паролів, розробити суворі правила для корпоративної парольної політики та контролювати їх виконання.

Рисунок 3.37 – Вразливості веб-додатків

На ресурсах мережевого периметра часто зберігаються у відкритому вигляді важливі дані, які допомагають зловмиснику розвинути атаку. Це можуть бути резервні копії веб-застосунків, конфігураційна інформація про систему, облікові дані для доступу до критично важливих ресурсів або ідентифікатори користувачів, до яких зловмисник може підібрати пароль.

Щоб запобігти виявленим загрозам, рекомендується переконатися, що у відкритому вигляді (наприклад, на сторінках веб-додатку) не зберігається чутлива інформація, що становить інтерес для зловмисника. До такої інформації можуть належати облікові дані для доступу до різних ресурсів, адресна книга компанії, що містить електронні адреси та доменні ідентифікатори співробітників тощо. Якщо у компанії не вистачає ресурсів, щоб виконати такі перевірки власними силами, то рекомендується залучати сторонніх експертів для тестування на проникнення Актуальною залишається проблема несвоєчасного оновлення ПЗ.

Далі подано результати внутрішніх тестів на проникнення. Для цього в усіх системах, що досліджуються, необхідний повний контроль над внутрішньою інфраструктурою. У середньому для цього потрібно чотири кроки.

Типовий вектор атаки будується на підборі словникових паролів та відновлення облікових записів з пам'яті ОС за допомогою спеціальних утиліт.

Повторюючи ці кроки, зловмисник переміщається в мережі від одного вузла до іншого аж до виявлення облікового запису адміністратора домену.

Рекомендується забезпечити захист інфраструктури від атак, спрямованих на відновлення облікових записів із пам'яті операційної системи. Для цього на всіх робочих станціях привілейованих користувачів, а також на всіх вузлах, до яких здійснюється підключення з використанням привілейованих облікових записів, встановити Windows версії вище 8.1 (на серверах — Windows Server 2012 R2 або вище) та включити привілейованих користувачів домену до групи Protected Users . Крім того, можна використовувати сучасні версії Windows 10 на робочих станціях та Windows Server 2016 на серверах, в яких реалізовано систему Remote Credential Guard, що дозволяє ізолювати та захистити системний процес lsass.exe від несанкціонованого доступу. Забезпечити додатковий захист привілейованих облікових записів (зокрема адміністраторів домену). Гарною практикою є використання двофакторної автентифікації.

Також як рекомендації пропонується відключити протоколи канального та мережевого рівня, що не використовуються в локальній обчислювальній мережі. Якщо ці протоколи потрібні для роботи будь-яких систем, слід виділити для них окремий сегмент мережі, до якого немає доступу з сегмента користувача.

КІС залишаються вразливими для атак зловмисників. З кожним роком збільшується частка компаній, де вдається отримати доступ до ресурсів внутрішньої мережі від зовнішнього зловмисника.

Використання методів соціальної інженерії [54] та експлуатація недоліків захисту бездротових мереж додатково підвищують шанси успішне подолання мережевого периметра. Як правило, вектори атак ґрунтуються на експлуатації відомих недоліків безпеки та здебільшого не вимагають від зловмисника глибоких технічних знань.

Для підтримки високого рівня захищеності системи необхідно дотримуватись загальних принципів та рекомендацій забезпечення інформаційної безпеки. Необхідно регулярно проводити аналіз захищеності

веб-додатків, при цьому найбільш ефективним методом перевірки є метод білої скриньки, що передбачає аналіз вихідного коду. В якості превентивної міри рекомендується використовувати міжмережевий екран рівня програм (web application firewall) для запобігання експлуатації вразливостей, які можуть з'являтися при внесенні змін до коду або додавання нових функцій.

У рамках тестування на проникнення дії експертів рідко виявляються службою безпеки компаній, а отже, і реальні зловмисники можуть тривалий час перебувати в інфраструктурі та залишатися непоміченими. Тому важливо не тільки захищати мережевий периметр, а й проводити регулярний ретроспективний аналіз мережі з метою виявити проникнення, що вже сталося. Для пошуку слідів компрометації рекомендується використовувати спеціалізовані засоби глибокого аналізу мережного трафіку, здатні виявляти складні цільові атаки як реальному часі, і у збережених копіях трафіку. Таке рішення дасть можливість не тільки побачити факти злому, а й відстежувати мережеві атаки, у тому числі запуск шкідливих утиліт, експлуатацію вразливостей КІС та атаки на контролер домену. Це дозволить зменшити час потайної присутності порушника в інфраструктурі і тим самим мінімізувати ризики витоку важливих даних та порушення роботи бізнес-систем, знизити можливі фінансові втрати. Додатково рекомендується використовувати спеціальне антивірусне програмне забезпечення, яке перевіряє файли в ізольованому середовищі, виявляє наявність шкідливих програм та допомагає блокувати шкідливу активність.

Важливо дотримуватися всіх рекомендацій у комплексі, оскільки навіть окремі прогалини у механізмах захисту можуть спричинити зло інфраструктуру та компрометацію критично важливих ресурсів.

Рекомендується регулярно проводити тестування на проникнення, щоб виявляти вектори атак на корпоративну систему та на практиці оцінювати ефективність вжитих заходів захисту.

ГЛАВА 4 АНАЛІЗ ЕФЕКТИВНОСТІ ЗАСТОСУВАННЯ КОМПЛЕКСНОГО ТЕСТУВАННЯ КОРПОРАТИВНИХ ІНФОРМАЦІЙНИХ СИСТЕМ

4.1 Оцінка обізнаності про рівень уразливості корпоративних інформаційних систем

На додаток до робіт з тестування на проникнення, важливо проводити перевірки обізнаності співробітників у питаннях інформаційної безпеки. Роботи виконуються за заздалегідь узгодженими сценаріями, які імітують реальну атаку зловмисника. Перевірки здійснюються шляхом телефонної взаємодії та розсилки електронних листів. У телефонній розмові робляться спроби дізнатися у користувачів ту чи іншу цінну інформацію. Електронні листи містять вкладені файли або посилання на веб-ресурс, де потрібно ввести облікові дані. Під час перевірки фіксується реакція співробітників: факти переходу за посиланням, введення облікових даних чи запуску вкладення.

За статистикою майже третина користувачів переходить за посиланням або запускає файл, а кожен десятий співробітник вводить свої облікові дані в фальшиву форму автентифікації. Помітна частка користувачів розкриває конфіденційну інформацію в розмові по телефону або листується з умовним зловмисником, повідомивши при цьому додаткові відомості про компанію або про себе.

З метою виявлення та запобігання атак методами соціальної інженерії рекомендується регулярно проводити навчання співробітників, спрямоване на підвищення їхньої компетенції у питаннях інформаційної безпеки, з контролем результатів.

Бездротові мережі [55] є потенційним вектором проникнення у внутрішню інфраструктуру підприємства.

Зловмиснику достатньо встановити на ноутбук загальнодоступне програмне забезпечення для атак на бездротові мережі та придбати недорогий

модем, який може працювати в режимі моніторингу трафіку. У семи із восьми протестованих систем бездротові мережі доступні за межами контрольованої зони, а це значить, що зловмисник може проводити атаки, просто перебуваючи на прилеглій території, наприклад, на парковці поряд з офісом або в кафе на цокольному поверсі будівлі. Майже у всіх мережах досліджуваних компаній використовується протокол WPA2 із методами автентифікації PSK чи EAP.

Залежно від методу аутентифікації, що використовується, перевіряється можливість реалізації різних типів атак. Для WPA2/PSK проводиться перехоплення рукостискання між точкою доступу та легітимним клієнтом точки доступу з наступним підбором паролів шляхом перебору. Успіх цієї атаки зумовлений складністю пароля. У ході перевірок було встановлено, що словникові ключі для підключення до бездротової мережі використовуються у половині систем. Інший спосіб атаки – створення підробленої точки доступу – застосовується для будь-якого методу аутентифікації. Якщо під час підключення до бездротової мережі не виконується автентифікація сертифікатів, зловмисник може створити підробну точку доступу з ідентичною назвою мережі (ESSID) і більш потужним сигналом, ніж у оригінальної. У разі підключення клієнта до цієї точки доступу зловмисник отримує його ідентифікатор у відкритому вигляді та значення NetNTLM v1 challenge-response, за допомогою якого може підібрати пароль методом перебору.

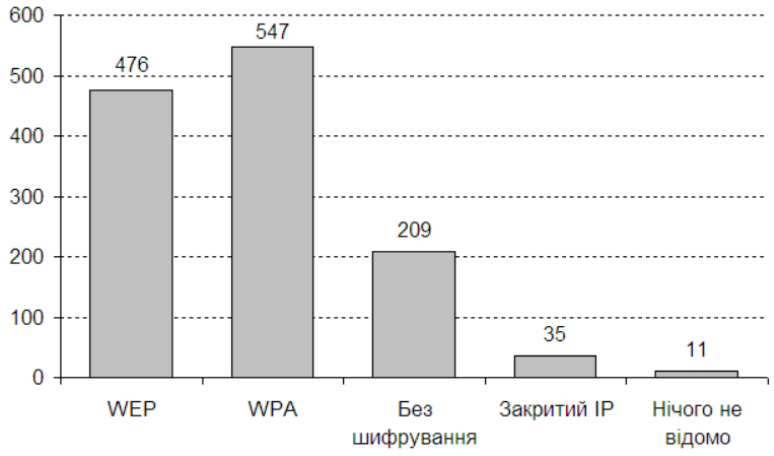

Рисунок 4.1 – Діагарама методів шифрування

Результат атаки залежить від того, наскільки співробітники компанії обізнані з інформаційною безпекою.

Рекомендується регулярно проводити аналіз захищеності бездротових мереж, щоб виявляти помилки конфігурації та потенційні вектори проникнення у внутрішню мережу.

4.2 Аналіз ефективності комплексного тестування КІС даними

Перед будь-яким інтеграційним процесом завжди стояло завдання покращення роботи комплексу інформаційних систем. Як критерії оцінки добре побудованого інтеграційного процесу можна виділити такі:

- швидкість передачі даних;
- ефективність обміну даними;
- оцінка користувачів інтеграційних систем.

Розглянемо використання t-критерію Стьюдента визначення наявності відмінностей між двома вибірками. При цьому вибірки можуть бути:

- незалежними, незв'язними з різним числом значень у вибірках;
- залежними, пов'язаними з рівним числом значень у вибірках.

Критерій t Стьюдента спрямовано оцінку відмінностей величин середніх x і y двох вибірок X і Y, які розподілені за нормальним законом. Однією з головних переваг критерію є широта його застосування.

Як правило, у дослідженнях найчастіше застосовуються такі параметричні критерії як: t-критерій Стьюдента, що дозволяє оцінювати відмінності середніх у двох вибірках і F - критерій Фішера, який оцінює різницю між двома дисперсіями. Вони дозволяють отримати найбільш наочне уявлення аналізованих сукупностей. Обчислення значення t здійснюється за формулою:

$$t = \frac{d}{Sd}, \tag{4.1}$$

$$d = \frac{\sum d_i}{n} = \frac{\sum (x_i - y_i)}{n}, \tag{4.2}$$

де $d_i = x_i - y_i$ - різниці між відповідними значеннями змінної X та змінної Y, d середнє цих різниць, n – обсяг вибірки, межі підсумовування від i = 0 до n.

У свою чергу Sd обчислюється за такою формулою:

$$Sd = \sqrt{\frac{\sum d_i^2 - \frac{(\sum d_i)^2}{n}}{n(n-1)}}, \tag{4.3}$$

Число ступенів свободи визначається за формулою k = n – 1. Для застосування t-критерію Стьюдента необхідно дотримуватися таких умов:

- вимір може бути проведений у шкалі інтервалів та відносин;
- порівнянні вибірки мають бути розподілені за нормальним законом.

Для розрахунку t-критерію Стьюдента необхідно скористатися процедурою «Парний двовибірковий t-тест для середніх» із пакета аналізу програмного продукту Microsoft Excel (рисунок 4.4).

Нульова гіпотеза H0 : μ1 - μ2 = δ приймається, якщо | t | < tкр1 (інакше відкидається); гіпотеза H0 при конкуруючій гіпотезі H1: μ1 > μ2 + δ приймається, якщо t < tкр2; при конкуруючій гіпотезі H1: μ1 < μ2 + δ нульова гіпотеза приймається під час виконання нерівності tкр2 < t.

	Змінна 1	Змінна 2
Середнє	3,27	2,22
Дисперсія	1,6267777778	1,166222222
Огляд	10	10
Кореляція Пірсона	0,817652538	
Гіпотетична різниця середніх	0	
df	9	
t-статистика	4,516158041	
P(T<=t) однобічне	0,000727399	
t критичне однобічне	1,845563454	
P(T<=t) двобічне	0,053454444	
t критичне двобічне	2,278848584	

Рисунок 4.2 - Результат розрахунку процедури «Парний двовибірковий t-тест для середніх»

Таким чином, початкове припущення підтвердилося, дійсно, середній час роботи інтеграційних операцій, після впровадження сценарію комплексного тестування КІС дає результати кращі порівняно з попередніми. На підставі статистичної оцінки інтеграційного експерименту можна констатувати, що виконана робота досить значуща та обґрунтована.

ВИСНОВКИ

У ході проведеного дослідження було розглянуто найбільш популярні методи тестування, застосовні щодо комплексного тестування корпоративних інформаційних систем. Показано, що комплексне тестування КІС відіграє важливу роль у процесі життєвого циклу ІС для визначення її працездатності та надійності.

Під час проведення досліджень у роботі отримано такі теоретичні та прикладні результати.

Під час аналізу теоретичних положень життєвого циклу тестування КІС були отримані концепції, необхідні для визначення поняття комплексного тестування та виділення його основних видів.

Для розробки тестів детально розглянуто методи функціональних діаграм та попарного тестування. Показано переваги проведення аналізу предметної галузі на основі онтологічної моделі.

На основі порівняльного аналізу методів тестування зроблено висновки, що для якіснішого проведення комплектного тестування КІС необхідне використання сукупності розглянутих методів. Для запропонованих методів визначено основні концепції та етапи проведення тестування.

В рамках навантажувального тестування розглянуто підхід, що ґрунтується на моделях. Показано його ефективне застосування для тестування прикладного програмного забезпечення в ІТ-середовищі функціонування, що включає СУБД, бази даних, системне програмне забезпечення, розгорнуті на обладнанні комплексу технічних засобів ІС.

Докладно досліджено технологію навантажувального тестування з використанням моделей, сформовано вимоги до проведення навантажувального експерименту та властивості об'єкта тестування, що охоплюють усі аспекти планування, проведення навантажувального експерименту та аналізу його. результатів. Показано, що використання моделей суттєво знижує

трудомісткість тесту навантаження і забезпечує можливість повторного використання підготовленого експерименту.

Розглянуто низку інструментальних засобів, що надають можливості для реалізації автоматизованого тестування. Кожен із розглянутих інструментів було оцінено з погляду його впровадження у існуючий процес розробки, і навіть з погляду задоволення вимог КІС. Наведено результати аналізу щодо основних із запропонованих інструментів. Показано, що автоматизація процесу тестування допомагає компаніям скорочувати час, що витрачається на тестування, а також спрощувати весь процес, оскільки застосовуються спеціальні програмні інструменти для створення та запуску тестів та перевірки результатів їх виконання.

З метою виявлення недоліків захисту різних компонентів системи та визначення потенційних атак на інформаційні ресурси розглянуто найбільш ефективні способи аналізу захищеності ІС. Надано низку рекомендацій щодо підвищення захищеності ІВ та інформаційної безпеки. Показано, що для підтримки високого рівня захищеності системи необхідно дотримуватись загальних принципів та рекомендацій забезпечення інформаційної безпеки. Необхідно регулярно проводити аналіз захищеності веб-додатків, при цьому найбільш ефективним методом перевірки є метод білої скриньки.

Таким чином, запропоновану методику (сценарій) комплексного тестування КІС можна використовувати для оцінки надійності та ефективності ІС, що функціонує у бізнес-середовищі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Мельникова К.С. Аналіз методологій тестування корпоративних систем. Theories, methods and practices of the latest technologies. Abstracts of III International Scientific and Practical Conference. Tokyo, Japan 2022. Pp. 91.

2. Beizer B. Software testing techniques. (Second edit.) International Thomson Computer Press, 1990. – 550.pp.

3. Boehm B. Software Engineering Economic Prentice-Hall,Inc, N.J. 1981. – 767 pp.

4. Лайза Кріспін, Джанет Грегорі. Гнучке тестування: практичний посібник для тестувальників ПЗ та гнучких команд. Agile Testing: A Practical Guide for Testers and Agile Teams. — М. : «Вільямс», 2010. — 464 с. — (Addison-Wesley Signature Series) — 1000 п.

5. Канер Кем, Фолк Джек, Нгуен Енг Кек. Тестування ПЗ. Фундаментальні концепції менеджменту бізнес-додатків. — Київ : ДіаСофт, 2001. — 544 с.

6. IEEE Guide to Software Engineering Body of Knowledge, SWEBOK, 2004

7. Myers G.J. The Art Of Software Testing [Text] / G.J. Myers — New York: John Wiley & Sons, Inc., 2004. — 254 p.

8. Основи інженерії якості програмних систем [Текст]: монографія / Ф.І. Андон, Г.І. Коваль, Т.М. Коротун, В.Ю. Суслов; НАН Украины — К. Академпериодика, 2002. — 502 с.

9. ДСТУ 2844-94. Програмні засоби ЕОМ. Забезпечення якості. Терміни та визначення [Текст]. — Введ. 1.08.1995. — К.: Держстандарт України, 1995. — 57 с.

10. ДСТУ 2850-94. Програмні засоби ЕОМ. Показники і методи оцінювання якості [Текст]. — К.: Держстандарт України, 1994. – 20 с.

11. ISO/IEC 9126. 2001. Software engineering – Software product quality –

Part 1: Quality model. Part 2: External metrics. Part 3: Internal metric. Part 4: Quality in use metrics [Text] — Geneva, Switzerland: International Organization for Standardization.

12. Patton R. Software Testing [Text] / R. Patton. — 2nd Edn. — Indianapolis: Sams, 2005. — 408p.

13. Burnstein I. Practical Software Testing. A process-oriented approach. Springer-Verlag, New York, 2003, – 732 p

14. Особливості тестування веб-додатків [Електронний ресурс] – Режим доступу до ресурсу: http://dynamic-design.com.ua/novosti/uk/brauzer-osoblivostitestuvanna-veb-dodatkiv/

15. QALight Club “Тестування веб-сервісів» [Електронний ресурс] – Режим доступу до ресурсу: https://dou.ua/calendar/28300/

16. What are web System? Architecture, Types, Example [Electronic resource] - Resource access mode: https://www.guru99.com/web-service-architecture.html

17. Ревенчук І.А. Методичні вказівки до лабораторних робіт з дисципліни «Якість та тестування программного забезпечення» для студентів усіх форм навчання напряму 7.05010302 – «Інженерія программного забезпечення» / І.А. Ревенчук, Т.С. Ткачова. – Харків: ХНУРЕ, 2010. 41с.

18. Considerations for Managing Internet of Things (IoT) Cybersecurity and Privacy Risks / K. Boeckl [та ін.]. — 06.2019. — Режим доступу до ресуру: https://doi.org/10.6028 /NIST.IR.8228

19. Пономаренко В. С. Інструментальні засоби розробки та підтримки баз даних розподілених інформаційних систем : навч. посібн. / В. С. Пономаренко, Л. А. Павленко. – Х. : Вид. ХДЕУ, 2001. – 132 с.

20. Пономаренко В. С. Організація даних у розподілених інформаційних системах : навч. посібн. / В. С. Пономаренко, Л. А. Павленко. – Х. : РІО ХДЕУ, 2000. – 104 с.

21. Norman, Alan T. Computer Hacking Beginners Guide: How to Hack Wireless Network, Basic Security and Penetration Testing, Kali Linux, Your First Hack. Independently published, 2018.

22. Johansen, Gerard, et al. Kali Linux 2–Assuring Security by Penetration Testing. Packt Publishing Ltd, 2016

23. BSI - Study A Penetration Testing Model. Federal Office for Information Security, 111 p. https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/Studies/Penetration/penetration_pdf.html

24. Oriyano Sean-Philip. Penetration Testing Essentials. Sybex, a Wiley brand, 2017, 363 p

25. Elfriede Dustin, Implementing Automated Software Testing: How to Save Time and Lower Costs While Raising Quality/ Elfriede Dustin, Thom Garrett, Bernie Gauf. Addison-Wesley Professional – 2009. - 368 p.

26. Gerard Meszaros, xUnit Test Patterns: Refactoring Test Code / Gerard Meszaros – 2007. – 833p.

27. Howard Wainer. Computerized Adaptive Testing: a primer. – London: Lawrence Erbaum Assocsates, Inc., 2000. – 335 p.

28. John Michael Linacre. Computer-Adaptive Testing: A Methodology Whose Time Has Come. – Seoul, South Korea: Komesa Press, 2000. – 58 p

29. John Michael Linacre. Computer-Adaptive Testing: A Methodology Whose Time Has Come. – Seoul, South Korea: Komesa Press, 2000. – 58 p

30. Говорущенко Т.О. Система повторного тестування програмного забезпечення // Радіоелектронні і комп'ютерні системи – Харків: НАУ "ХАІ", 2005. - №4, с.120 – 126

31. Berman O., Ashrafi N., Optimization models for reliability of modular software systems // IEEE Trans. Software Reliability. – 1993. V. 19. – P. 1119 – 1123.

32. Коротун Т.М. Моделі та методи інженерії тестування програмних систем в умовах обмежених ресурсів. Дис. канд. фіз.-мат. наук. Інститут кібернетики ім. В.М. Глушкова НАН України, Київ, 2005. – Держ. обл. № 0405U003083.

33. ДСТУ 2853-94 «Програмні засоби ЕОМ. Підготовлення і проведення випробувань». – К.: Держстандарт України, 2000. – 29 с.

34. Gelperin D., Hetzel B. The Growth of Software Testing // Commum. ACM. – 1988. – V. 31, N 6, – P. 687 – 695.

35. Розробка методик, програмних засобів тестування та оцінки надійності програмного забезпечення // НДР 6.1.3./1702- 92, N UA0I009618P, 1992. – 1994.

36. Musa J.D., Everett W.W. Software-Reliability Engineering: Technology for the 1900 // IEEE Software. – 1990. – № 11. – P. 36 – 43.

37. Quercia V., Spainhour S. WebMaster in a Nutshell: A Desktop Quick Reference. Third edition. – O`Reilly Media, 2002, 520 pp.

38. Yacoub S. M., Cukic B., and Ammar H. H. A component-based approach to reliability analysis of distributed systems // Proc. 18th IEEE Symp. Reliable Distributed Syst. – 1999. – P. 158 – 167.

39. Software testing training and software testing services. [Електронний ресурс]. Режим доступу до ресурсу: http://www.rbcs-us.com/

40. Apache Software Foundation. [Електронний ресурс]. - Режим доступу до ресурсу: http://jmeter.apache.org/

41. Stack Overflow [Електронний ресурс] – Режим доступу до ресурсу: http://stackoverflow.com

42. Шилдт Г. Java 8. Повне керівництво. – К. : Діалетіка, 2015. – 1376 с.

43. Савін Р. Тестування Дот Ком. М:, «Діло», 2007.

44. Що таке чек-лист та як працювати з ними [Електронний ресурс]. – Режим доступу до ресурсу: https://training.qatestlab.com/blog/technical-articles/work-withchecklist/

45. Що таке тест-кейс [Електронний ресурс]. – Режим доступу до ресурсу: https://qalight.ua/baza-znaniy/test-case-2/

46. Документація Locust [Електроний ресурс]. – Режим доступу до ресурсу: https://docs.locust.io/en/stable/

47. Що таке vps хостинг? [Електроний ресурс]. – Режим доступу до ресурсу: https://hostiq.ua/ukr/info/what-is-vps/

48. NoSQL Databases / C. Strauch. // Lecture, Stuttgrat Media University

49. Scalable SQL and NoSQL data stores. / Cattell. // Acm Sigmod Record. – 2011. – №39. – С. 12–27.

50. An introduction to Graph Data Management. In Graph Data Management / R. Angles, C. Gutierrez. – 2018. – С. 32

51. Citrus Frameworkhttps. [Електроний ресурс]. – Режим доступу до ресурсу: https://citrusframework.org

52. API Wizard [Електроний ресурс]. – Режим доступу до ресурсу: https://www.api-wizard.com.

53. Кібератаки України 2022. [Електроний ресурс]. – Режим доступу до ресурсу: https://www.ukrinform.ua/tag-kiberataka

54. A Survey and Evaluation of the Potentials of Distributed Ledger Technology for Peer-to-Peer Transactive Energy Exchanges in Local Energy Markets / Siano, De Marco, Rolán, Loia. // IEEE Systems Journal. – 2019.

55. Effects of storage heterogeneity in distributed cache systems. / K. S. Reddy, S. Moharir, N. Karamchandani. // In 2018 16th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt). IEEE. – 2018. – С. 1–8.