Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:
«Засоби тестування відмовостійкості елементів комп'ютерної системи»

Виконав: студент 2 курсу, групи 2КІ-21м
спеціальності 123 Комп'ютерна інженерія
Луценко Ю.В.

Керівник к.т.н., доц. каф. ОТ
_____ Колесник І.С..

Опонент к.т.н. доц. каф. МБІС
_____Карпінець В.В.

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О.Д.

«22» 12 2022 р.

Вінниця 2022

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Освітньо-кваліфікаційний рівень магістр
Спеціальність 123 Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри
обчислювальної техніки
проф., д.т.н. О. Д. Азаров

«15» 09 2022 р.

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Луценку Юрію Вікторовичу

1 Тема роботи «Засоби тестування відмовостійкості елементів комп'ютерної системи», керівник роботи Колесник Ірина Сергіївна, к. т. н., доцент кафедри ОТ,затверджені наказом вищого навчального закладу від 15.09.2022 року №205-А.

2 Строк подання студентом роботи 15.09.2022 р.

3 Вихідні дані до роботи: середовища розробки: MicrosoftWindowsServer 2008 R2 x64, маршрутизатор AsusRT-AC68U; Сервер XeonE5-2630, 12 x 2.60 ГГц, процесорів 2 оперативна пам'ять – 8 ГБ, пам'ять – 2 x 500 ГБ, зовнішнє сховище даних BLOB у SharePointFoundation.

4 Зміст розрахунково-пояснювальної записки: вступ; теоретичні основи тестування комп'ютерних систем; аналіз засобів проведення тестування комп.терної системи; апробація засобів тестування відмовостійкості комп'ютерної системи; аналіз ефективності застосування; висновки; перелік посилань; додатки.

5 Перелік графічного матеріалу: життєвий цикл тестування; загальна схема основних видів тестування;алгоритм процедури тестування за сценарієм 1.

6 Консультантів розділів роботипредставленов табл.1.

Таблиця 1— Консультантирозділівроботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1,2,3,4	Колесник І.С., к.т.н.,, доц. каф. ОТ		
5	Небава М.І., к.е.н., професор	Небава	Небава

7 Дата видачі завдання 15.09.2022 р.

8 Календарнийпланнаведенов табл.2.

Таблиця 2—Календарний план

№ з/п	Назва етапів виконання магістерської роботи	Строк виконання етапів роботи	Примітка
1	Постановка задач роботи	15.09.22	вик.
2	Методологія тестування комп'ютерних систем	15.09-16.09.22	вик.
3	Засоби комплексного тестування комп'ютерної системи	16.09-24.09.22	вик.
4	Тестування комп'ютерної системи та її продуктивності	24.09-04.10.22	вик.
5	Тестування навантаження з використанням різних інструментальних засобів	04.10-14.10.22	вик.
6	Аналітичні показники тестування безпеки	14.10-23.10.22	вик.
7	Оцінка обізнаності про рівень уразливості комп'ютерної системи	24.10-31.10.22	вик.
8	Аналіз ефективності тестування комп'ютерної системи	01.11-16.11.22	вик.
9	Розрахунок економічної частини роботи	17.11-30.11.22	вик.
10	Оформлення пояснювальної записки та ілюстративного матеріалу	01.12-06.12.22	вик.
11	Аналіз виконання роботи, висновки, додатки	07.12-06.12.22	вик.
12	Перевірка якості виконання магістерської роботи та усунення недоліків	15.12.21	вик.

Студент _____ Луценко Ю.В.

Керівник роботи _____ Колесник І.С.

АНОТАЦІЯ

УДК 004.93

Луценко Ю.В. Засоби тестування відмовостійкості елементів комп'ютерної системи. Магістерська кваліфікаційна роота зі спеціальності 123 – компютерна інженерія, освітня програма комп'ютерна інженерія. Вінниця, ВНТУ 2022, 137 с.

На укр.мові. Бібліогр.: 40 назв., рис.45, табл. 15.

Магістерська кваліфікаційна робота присвячена аналізові засобів тестування відмовостійкості та розробці сценарію комплексного тестування для підвищення продуктивності та безпеки комп'ютерних систем.

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

В роботі розглянуто теоретичні засади тестування комп'ютерних систем та проаналізовано з урахуванням розробленої теорії практичні методи тестування відмовостійкості їх програмного забезпечення. Крім того розроблено сценарій комплексного тестування відмовостійкості,, його ефективність підтверджено експериментальним шляхом.

Ключові слова: тестування, комп'ютерна система, модель, відмовостійкість, надійність.

ANNOTATION

Lytsenko Y.V. Means of testing the fault tolerance of computer system elements. Master's qualification route in the specialty 123 — computer engineering, educational program computer engineering. Vinnitsa, VTNU, 2022, 137 p.

In the Ukr. leng. Libr. name 40, figure 45, table 15.

The master's thesis is devoted to the analysis of fault tolerance testing tools and the development of a comprehensive testing scenario to improve the performance and security of computer systems.

Analysis of research and publications in the field of fault tolerance testing showed that consideration of such aspects of testing as technical, economic, and psychological is of significant practical interest.

The paper examines the theoretical principles of testing computer systems and analyzes the practical methods of testing the fault tolerance of their software taking into account the developed theory. In addition, a comprehensive fault tolerance testing scenario was developed, its effectiveness was confirmed experimentally.

ВСТУП

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

Аналіз публікацій показав, що сенс терміна «тестування» та визначення його передається різними авторами як: «тестування є процесом, що демонструє відсутність помилок у програмі» [26], «мета тестування – показати, що програма коректно виконує передбачені функції» [21], «Тестування — це процес, що дозволяє переконатися в тому, що програма виконує своє призначення» [31] і є недостатньо коректними з точки зору економічної складової (тобто ціни розробки програмного продукту (ПП)). У свою чергу, відмовостійкість — це здатність КС продовжувати дії, задані програмою, після виникнення несправностей. Введення відмовостійкості вимагає надлишкового апаратного та програмного забезпечення. Напрями, пов'язані із запобіганням несправностей і з відмовостійкістю основні для забезпечення надійності. Найчастіше результати своєї роботи програміст неспроможний оцінити з погляду ефективності КС, де вони використовуються. Він практично нічого не може сказати про те, наскільки повно протестований код програми. Крім того, дуже часто розробникам, менеджерам проектів, керівнику фірми необхідно вирішувати питання, пов'язані із скороченням витрат на виробництво ПП та підвищенням якості програмного забезпечення. Основним способом вирішення цих проблем є тестування відмово стійкості програмного забезпечення КС, так і комп’ютерної системи в цілому. Процес тестування включає вирішення питань не тільки технічного характеру (організація ефективного процесу тестування, визначення часу тестування, використання або невикористання інструментальних засобів і т.д.), але і питань економічного і психологічного характеру. Безумовно, виходячи з вищевикладеного, питання, які розглядаються у роботі, на сьогоднішній день є актуальними.

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

Відповідно до вищесказаного, метою є дослідження тестів для перевірки відмовостійкості програмних додатків КС з точки зору психологічного, економічного та технічного підходів.

Метою роботи є аналіз засобів тестування відмовостійкості та розробка сценарію комплексного тестування для підвищення продуктивності та безпеки комп’ютерних систем.

Позначена мета визначила об'єкт та предмет дослідження.

Об'єкт дослідження – засоби комплексного тестування відмовостійкості елементів комп’ютерних систем.

Предмет дослідження – розробка сценарію комплексного тестування відмовостійкості для підвищення продуктивності та надійності елементів комп’ютерних систем.

Для того, щоб досягти мети, необхідно вирішити **завдання**:

— розглянути теоретичні засади тестування комп’ютерних систем, що дозволяють здійснювати аналіз наявних методів тестування відмовостійкості елементів комп’ютерних систем;

— проаналізувати з урахуванням розробленої теорії практичні методи тестування відмовостійкості програмного забезпечення комп’ютерних систем;

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

— підтвердити шляхом експерименту ефективність реалізованого сценарію та його результати.

Наукова новизна дослідження полягає в тому, що в ньому визначено основні критерії ефективності тестування відмовостійкості, які застосовуються для оцінки продуктивності та надійності КС; розроблений сценарій комплексного тестування КС, що дозволяє на основі визначення типів помилок, які найчастіше

зустрічаються, задавати якісні тести, що забезпечують мінімізацію всіляких втрат у бізнес-середовищі організації.

Практична значимість дослідження полягає у розробці теоретичних положень концепції аналізу та порівняння засобів тестування відмовостійкості КС, що дозволяють здійснювати вибір найбільш відповідного засобу для практичного використання, покладеного в основу розробки сценарію комплексного тестування, що дозволить суттєво скоротити витрати на тестування відмовостійкості КС та підвищити якість її функціонування.

1 ТЕОРЕТИЧНІ ОСНОВИ ТЕСТУВАННЯ КОМП'ЮТЕРНИХ СИСТЕМ

1.1 Життєвий цикл тестування комп'ютерної системи

Комп'ютерні системи (КС) з погляду системного аналізу відносяться до складних систем, оскільки вони побудовані з різних підсистем: серверів БД, серверів додатків, кешуючих серверів, балансувальників, серверів резервного копіювання, безпеки і т.д. [18]. У добре спроектованій системі великий ступінь взаємної інтеграції, яка збільшує ризик виходу з ладу однієї або кількох підсистем КС, за яким слідує з ладу інших раніше працездатних підсистем і так далі аж до повного руйнування всієї КС.

У зв'язку з цим розробка методик (сценаріїв) тестування КС, що проводиться на всіх етапах її життєвого циклу, є актуальною [8]. 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:

— аналіз вимог (табл. 1.1) — на цьому етапі команда забезпечення якості (QA) розуміє вимоги з точки зору того, що тестуватимемо, і з'ясуємо вимоги, що тестуються; вимоги можуть бути функціональними або нефункціональними, такими, як продуктивність, тестування безпеки [10];

— планування випробувань (табл. 1.2) — цьому етапі визначаються зусилля та оцінки витрат для проекту;

— розробка тестового прикладу (таб. 1.3) — цьому етапі STLC команда тестування записує докладні тестові приклади, поряд із тестовими наборами команда тестування також готує тестові дані, якщо такі необхідні тестування;

— налаштування тестового середовища (табл. 1.4), яке визначає умови тестування програмного забезпечення.

Таблиця 1.1 — Критерії входу та результату етапу «Аналіз вимог»

Критерії входу	Заходи	Результат
1. Наступні документи повинні бути доступні: —технічні вимоги. —додатки архітектурні. 2. Поряд з вище зазначеними документами критерії приймання повинні бути чітко визначено.	1. Підготуйте список запитань чи запитів та отримайте відповіді від Business Analyst, System Architecture, Client, Technical Manager/Lead тощо. Складіть список того, що виконували всі типи тестів, такі як функціональність, безпека, продуктивність тощо. 2. Визначте напрямок та пріоритети тестування. 3. Перерахуйте деталі середовища тестування, де проводитимуться дії з тестування. 4. При необхідності перевірте техніко-економічне обґрунтування автоматизації та підготуйте техніко-економічне обґрунтування автоматизації.	Перелік питань з усіма відповідями, які повинні бути вирішені з бізнесу, тобто тестовані вимоги ТЕО автоматизації (якщо застосовується).

Виконання тесту (табл. 1.5) — 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, після усунення дефектів, ті ж випробування з помилками або

блокування можуть бути виконані знову для повторного тестування функціональності [10].

Таблиця 1.2 — Критерії входу та результату етапу «Планування випробувань»

Критерії входу	Заходи	Результат
1. Документи з вимогами (оновлена версія неясної чи відсутньої вимоги). 2. Автоматизація техніко-економічного обґрунтування.	1. Визначити мету та обсяг проекту. 2. Перерахуйте типи тестування, що у STLC. 3. Оцінка зусиль по тестуванню і плануванню ресурсів. 4. Вибір інструменту тестування, якщо потрібно. 5. Визначте огляд процесу тестування. Визначте середовище тестування, необхідне для всього проекту. 6. Підготуйте розклад випробувань. Визначте контрольні процедури. Визначте критерії входу, критерії відновлення і критерії виходу. 7. Визначте ризик, якщо є.	1. План тестування або документ про стратегію тестування. 2. Документ про оцінку зусиль з тестування.

Таблиця 1.3 — Критерії входу та результату етапу «Розробка тестового прикладу»

Критерії входу	Заходи	Результат
1.Документи з вимогами (оновлена версія незрозумілої або відсутньої вимоги). 2.Автоматизація техніко-економічного обґрунтування.	1.Підготовка тестових випадків. 2.Підготовка сценаріїв автоматизації тестування. 3.Підготовка даних.	1.Тестові випадки. 2.Тестові дані. 3.Тестування сценаріїв автоматизації.

Таблиця 1.4 — Критерії входу та результату етапу «Налаштування тестового середовища»

Критерії входу	Заходи	Результат
1. План випробувань доступний. 2. Тести доступні. 3. Тестові дані доступні.	1. Проаналізуйте вимоги та підготуйте список програмного та апаратного забезпечення, необхідного для налаштування тестового середовища. 2. Налаштуйте тестове середовище. 3. Після налаштування тестового середовища виконайте контрольні тести, щоб перевірити готовність середовища тестування.	1. Тестове середовище буде готове з тестовими даними. 2. Результати тестів.

Таблиця 1.5 — Критерії входу та результату етапу «Виконання тесту»

Критерії входу	Заходи	Результат
1. План тестування або документ про стратегії тестування. 2. Тестові випадки. 3. Тестові дані.	1. На основі планування тестування виконайте контрольні приклади. 2. Позначте стан тестових випадків, таких як Пройдено, Збій, Заблоковано, Не виконано тощо. Призначте ідентифікатор помилки для всіх невдалих та заблокованих тестових випадків. 3. Зробіть повторне тестування, коли дефекти усунуті. 4. Слідкуйте за дефектами до закриття.	1. Звіт про 2. виконанні тесту. 3. Звіт про дефекти.

Закриття циклу випробувань (табл. 1.6) — цьому етапі аналізуються помилки, розподіляються дефекти по типу і складності.

Таблиця 1.6 — Критерії входу та результату етапу «Закриття циклу випробувань»

Критерії входу	Заходи	Результат
1. Виконання тестового прикладу завершено 2. Звіт про виконаннятесту 3. Звіт про дефекти	1. Оцініть критерії завершення циклу на основі охоплення тестування, якості, вартості, часу, критичних бізнес-цілей та програмного забезпечення. 2. Підготуйте показники тесту на основі вказаних вище параметрів. 3. Підготувати звіт про закриття тесту	1. Звіт про закриття тесту 2. Тест метрики

	4. Поділитись передовим досвідом для будь-яких подібних проектів.	

Сценарії тестування STLC [18] повинні сприяти пошуку непередбачених помилок/проблем у роботі КС на всіх етапах життєвого циклу, що виражається замкненою послідовністю дій (рис. 1.1):

Рисунок 1.1 — Життєвий цикл тестування

— стадія 1 — загальне планування та аналіз вимог — 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 — для розробки, перегляду, уточнення, доопрацювання, переробки тощо. з тест-кейсами/тестовими сценаріями;

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

— стадія 6 — фіксація знайдених дефектів — для формування розуміння проблеми та уточнення важливості та терміновості проведення тестування;

— стадія 7 — аналіз результатів тестування — для обробки отриманих результатів, щоб приймати рішення про ще одну ітерацію;

— стадія 8 — звітність — для фіксування проміжних та кінцевих варіантів тестування.

Кожна з виділених стадій тестування може розглядатися як така, що забезпечує створення певного проміжного продукту — функціональної групи програм чи програмного засобу з деякими обмеженими характеристиками якості. Ці характеристики виділяються та деталізуються на основі первинного технічного завдання та специфікації вимог на ПЗ. У процесі проектування ПЗ вони уточнюються та конкретизуються у специфікаціях вимог на групи програм та їх компоненти [20]. 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 КС, можна з упевненістю стверджувати про працездатність програмного продукту.

1.2 Принципи та основні етапи комплексного тестування комп’ютерної системи

Тестування КС охоплює основні стадії життєвого циклу, аналогічний послідовності процесів розробки програмного забезпечення: постановка задачі для тесту, проектування, написання тестів, тестування тестів, виконання тестів та вивчення результатів тестування [22]. Щоб орієнтуватися у тестах, варто розглянути два основні підходи.

Перший підхід орієнтований на вивчення логіки програмного забезпечення КС 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 [14]. При цьому зовсім (або майже зовсім) не цікавляться специфікаціями.

Другий підхід тестування ґрунтується на виконанні фундаментального принципу функціонування КС: 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 [16]. 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 КС — це не тільки програмні компоненти, що використовуються в її складі, а й апаратне та організаційне забезпечення, то і в результатах її випробувань повинні 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 КС складається з наступних етапів [33]:

1) детальне вивчення проекту системи і його експлуатаційних документів;

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

3) безпосереднє тестування роботи КС, яке включає;

— тестування роботи всіх модулів КС;

— перевірка внутрішньосистемних зв'язків;

— тестування обладнання, у тому числі на наявність необхідних ліцензій;

— перевірка програмного забезпечення, у тому числі перевірка коду;

— тестування продуктивності та максимальних навантажень КС;

— тестування на відмови: несподівані програмні збої, вихід з ладу модулів системи, людський фактор тощо;

— тестування на захищеність КС — включає комплекс досліджень з виявлення способів злому системи та витоків інформації;

— загальна перевірка роботи системи;

4) тестування роботи системи керування ІТ-структурою;

5) тестування персоналу;

6) аналіз отриманих даних та вироблення стратегії з виправлення виявлених помилок.

Комплексне тестування призначене для тестування всіх функцій повністю зібраної системи. Воно спрямовано пошук невідповідності системи її вихідним цілям [34]. Таким чином, у комплексному тестуванні беру участь КС, опис її цілей та вся документація, яка поставлятиметься разом із системою.

1.3 Методологія тестування комп’ютерних систем

Методологія тестування є стратегією та підходом для тестування КС, щоб гарантувати, що програмний продукт придатний для експлуатації.

Методики тестування включають тестування того, що КС 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 [27].

Методології тестування КС — це різні підходи та способи забезпечення того, щоб КС була повністю протестована. Методології тестування охоплюють усі: від модульного тестування окремих модулів, інтеграційного тестування всієї КС до спеціалізованих форм тестування, таких як безпека та продуктивність [11].

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

На рис. 1.2 представлено загальну схему основних видів тестування КС, передбачених методологіями тестування.

Рисунок 1.2 — Загальна схема основних видів тестування

Згідно з цією схемою методологія тестування включає [36] функціональне тестування виконується з використанням функціональних специфікацій, наданих клієнтом, або з використанням специфікацій проектування, таких як сценарії використання, які надає команда розробників.

Функціональне тестування в методології тестування розбите на чотири компоненти: модульне тестування, інтеграційне тестування, системне тестування та приймальне тестування.

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 [19]. Кожна функція модуля тестується спеціальним модульним тестовим приладом, написаним тією ж мовою програмування, що сам модуль (рис. 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 — Фрагмент модульного тесту

Інтеграційне тестування — 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 (рис. 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.

Рисунок 1.4 — Демонстрація роботи системного тестування

Приймальне тестування (рис. 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, щоб переконатися, що вона працює належним чином і відповідає всім пред'явленим вимогам [38].

Рисунок 1.5 — Демонстрація роботи приймального тестування системи

Нефункціональне тестування включає тестування додатка на відповідність нефункціональним вимогам, які зазвичай включають вимірювання / тестування додатка на відповідність певним технічним якостям («здібностям»), наприклад: вразливість, масштабованість, зручність використання [11, 24].

У більшості методологій тестування існує кілька різних типів тестування продуктивності (рис. 1.6), наприклад [27]:

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

— навантажувальне тестування — перевірка того, що система може працювати у потрібному режимі, а час відгуку, коли він піддається очікуваному навантаженню;

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

Рисунок 1.6 — Демонстрація роботи тестування продуктивності

Тестування сумісності (рис. 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 [31], щоб переконатися, що програмний продукт працює, як очікується, у всіх різних комбінацій апаратних/програмних, і що всі функції послідовно підтримуються.

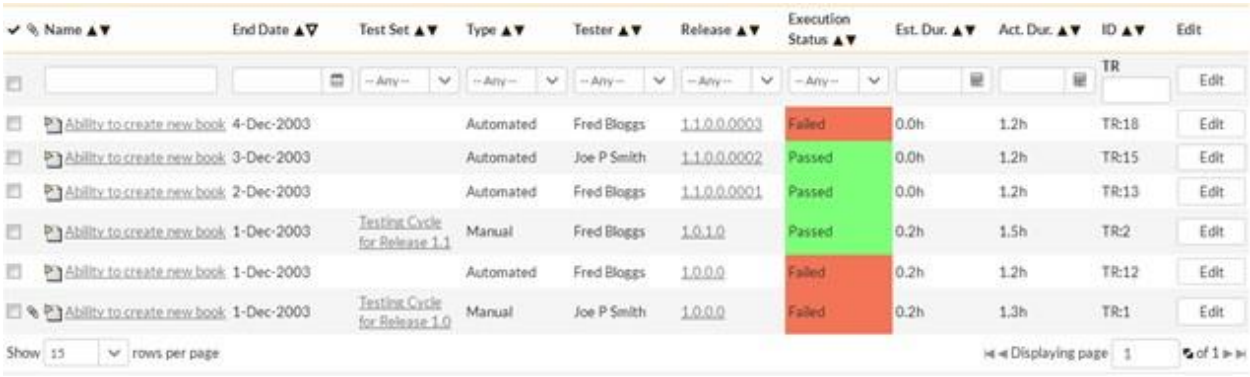


Рисунок 1.7 — Демонстрація роботи тестування сумісності

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.

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), яке 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 користувачам і складаються з трьох основних компонентів [37]:

— власне сценарія комплексного тестування з вказівкою дій, які мають бути досконалі під час виконання тесту;

— вхідних даних;

— очікуваних вихідних даних.

У комплексному тестуванні крім фахівців можуть брати участь і користувачі КС, ґрунтуючись на одному з методів [42]:

— досвідчена експлуатація, коли система тестується на робочому місці — це дозволяє побачити КС у роботі до того, як її почнуть експлуатувати;

— використання КС у створенні виробника для внутрішніх потреб, дозволяє усунути безліч помилок, але не дає змоги побачити деякі помилки інтеграції.

Таким чином, комплексне тестування КС 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 відмовостійкості комп’ютерної системи

Компонентами комплексного тесту є вихідні цілі, 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 та публікаціями про неї [43]. Виділяють такі підвиди комплексного тестування:

— тестування стресів або тестування з навантаженням — це спроба піддати систему максимальному «тиску» (наприклад, спробу одночасно підключити до системи поділу часу 100 терміналів);

— тестування обсягу передбачає спробу пред'явити системі великі обсяги даних протягом тривалого часу, щоб визначити відповідність кількості даних і даних, зазначених у специфікації;

— тестування конфігурації — це спроба перевірити роботи мінімальної та максимальної конфігурації системи з будь-якою апаратною платформою або програмою, з якою КС буде взаємодіяти;

— тестування сумісності. — спроба знайти несумісності нових та старих версій КС, при цьому взаємодія користувачів з попередньою версією має повністю зберегтися;

— тестування захисту — спроба порушити таємність у системі;

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

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

— тестування процесів налаштування системи — спроба визначити якість та ергономічність роботи системи;

— тестування надійності/готовності — спроба довести, що система не задовольняє вихідним вимогам надійності (середній час між відмовами, кількість помилок, здатність до виявлення, виправлення помилок та стійкість до помилок);

— тестування засобів відновлення — спроба перевірити здатність системи до відновлення (особливо у сфері роботи операційної системи, СУБД, систем передачі);

— 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;

— тестування публікацій — спроба повірити точність усієї документації;

— тестування психологічних факторів — спроба усунення дрібних недоліків; мінімізація незручності використання;

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

— 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 КС 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. При тестуванні 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.

У ході аналізу теоретичних основ комплексного тестування КС та аналізу його основних видів було зроблено висновок про те, що комплексне тестування КС не зводиться до перевірки її функції на відмовостійкість, а орієнтовано лише на оцінку якості та продуктивності. Для проведення комплексного тестування

відмовостійкості має бути розроблений сценарій, який має передбачити всі можливі варіанти, що забезпечують нестабільну роботу КС.

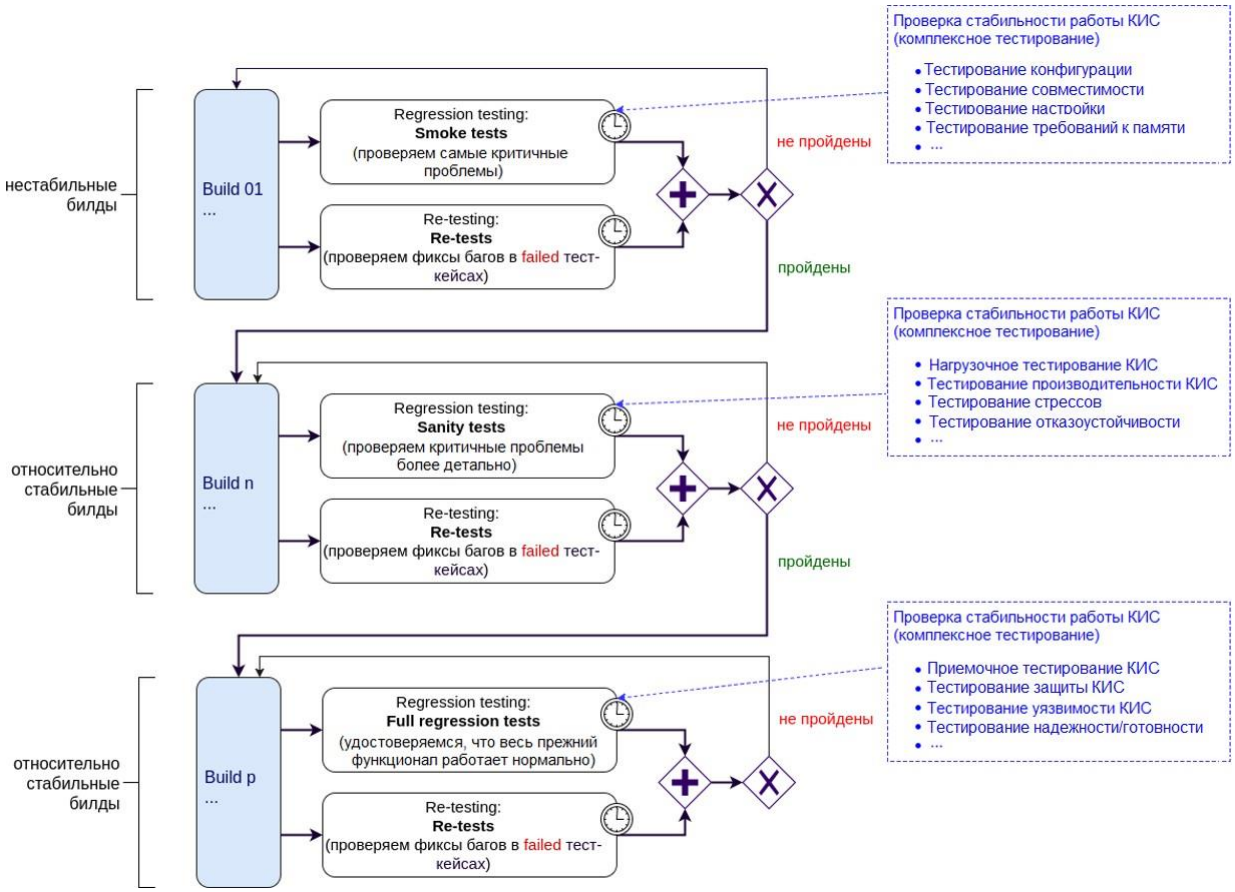


Рисунок 1.8 — Сценарій комплексного тестування КС

2 АНАЛІЗ ЗАСОБІВ ПРОВЕДЕННЯ ТЕСТУВАННЯ КОМП'ЮТЕРНОЇ СИСТЕМИ

2.1 Основні критерії ефективності засобів тестування комп'ютерної системи

Під тестуванням розуміють процес виконання програм на кінцевій множині вхідних даних X, отримання відгуку Y та його порівняння з еталонним безліччю вихідних значень Yет, з метою виявлення помилок та дефектів в ІВ [40]. Пара $(x, y_{ет}): x \in X, y_{ет} \in Y_{ет}$ називається тестовим випадком (тест-кейс), проте тестові випадки, згруповані за певною ознакою, іменуються тестовим комплектом.

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

$$\exists i, y^i_{ет} \in Y_{ет}, y_i \in Y: y^i_{ет} \neq y_i \qquad (2.1)$$

Якщо ж відрізняються закони розподілу вихідних даних (рис. 2.1), то в обох випадках вважається, що тестування пройшло успішно, оскільки виявлено як мінімум одну помилку.

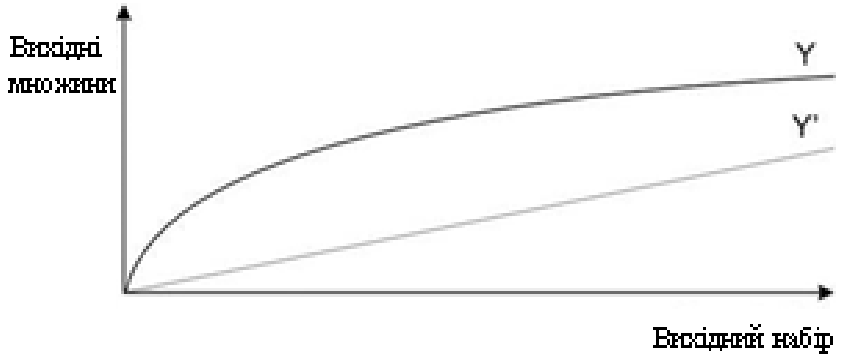


Рисунок 2.1 — Графіки залежностей результуючих та еталонних вихідних даних від вхідного комплекту

Інтенсивність виявлення помилок на одиницю витрат і надійність 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 якості КС призводить до використання різних способів тестування в процесі створення ПЗ. На рисунку 2.2 (блок В) наведено витратний компонент тестування залежно від удосконалення інструментарію, що застосовується, та методів тестування.

На практиці відомі наступні способи тестування, упорядковані за їх застосуванням витрат [43]:

— 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 (блок С) зіставлена на рисунку з витратами. Графік вказує, що з часом, у міру виявлення найбільш серйозних помилок і недоліків, ефективність низьковитратних методів падає разом з кількістю помилок, що виявляються.

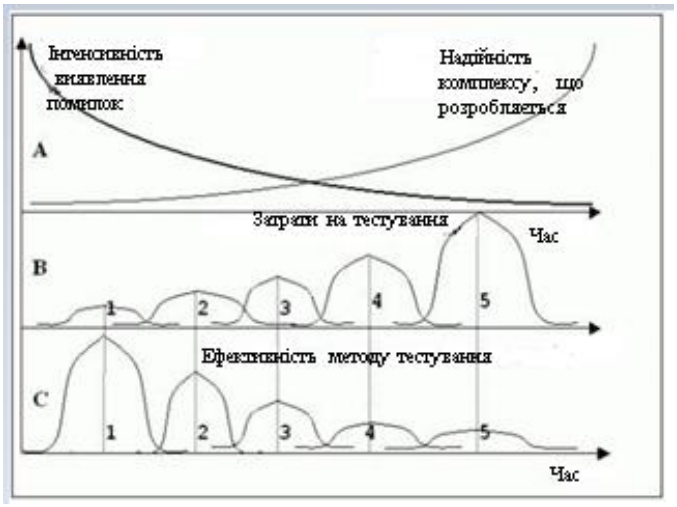


Рисунок 2.2 — Аналіз ефективності критеріїв тестування у процесі створення програмного продукту

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 [27]:

Структурні критерії, що використовують модель програми у вигляді "білої скриньки", що передбачає знання вихідного тексту програми або специфікації програми у вигляді потокового графа управління. До них відносяться:

— тестування команд (критерій C0);

— тестування гілок (критерій C1);

— тестування шляхів (критерій C2).

Функціональні критерії, що використовують модель «чорної скриньки», що передбачає формулювання в описі вимог до програмного виробу та забезпечення контролю ступеня виконання вимог замовника у програмному продукті. До них відносяться:

— тестування пунктів специфікації;

— тестування класів вхідних даних;

— тестування правил;

— тестування класів вихідних даних;

— тестування функцій;

— комбіновані критерії для програм та специфікацій.

Критерії стохастичного тестування, які формулюються в термінах перевірки наявності заданих властивостей у програми, 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 Хі-квадрат (x2) тощо);

— метод оцінки швидкості виявлення помилок — 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.

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.

2.2Засоби комплексного тестування комп'ютерної системи

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

Виділимо серед методів тестування: методи функціональних діаграм та попарного тестування [32, 36], 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 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 чи перетворення програми (дія, яку вхідна умова надає стан програми);

— причини та наслідки перетворюються 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.

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.

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 разів.

Для тестування з використанням ортогональних масивів слід виконати такі кроки:

— встановити комбінації змінних для вхідних даних;

— визначити значення, які можуть набувати змінних;

— 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 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 показано залежність

збільшення числа вичерпних та парних тестів від кількості тестових рівнів (можливої кількості значень параметрів).

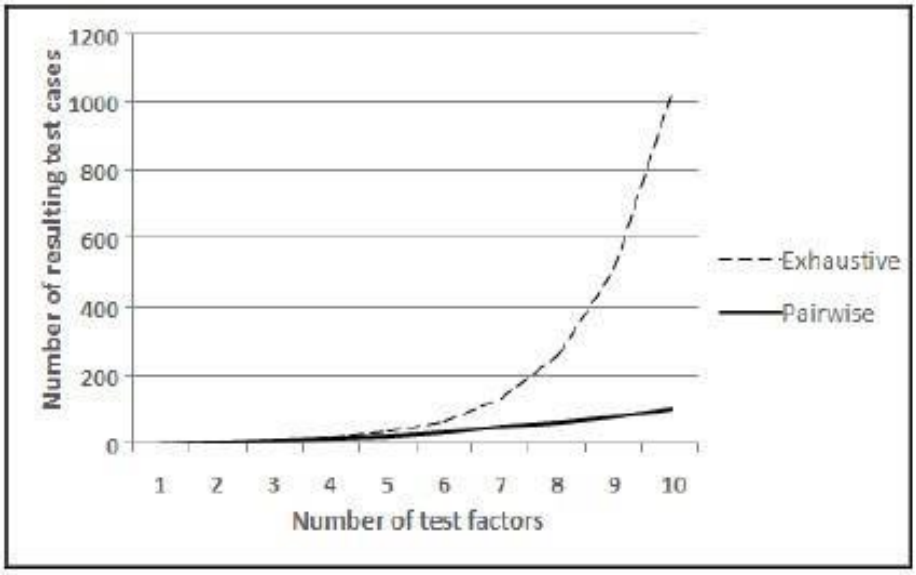


Рисунок 2.3 – Залежність збільшення кількості вичерпних та парних тестів від кількості тестових рівнів

Парне тестування — метод проектування чорної скриньки, у якому тестові випадки призначені виконання всіх можливих дискретних комбінацій кожної пари вхідних параметрів.

Вихідні дані програми залежать від багатьох факторів, наприклад, від вхідних параметрів, змінних стану та конфігурації середовища. Парне тестування використовує: аналіз граничних значень та розподіл еквівалентності, які припускають, що кожної пари вхідних параметрів системи повинні існувати всі можливі дискретні комбінації цих параметрів.

Формально стратегія парного тестування визначається наступним чином: дано набір з N незалежних випробувань факторів f1,f2, ..., fN, де кожен фактор fi має Li можливих рівнів fi = {li,1, ..., li, Li}, і набір тестів R. Кожен тест R містить N тест-рівень, по одному для кожного тест-фактора fi, i, в сукупності, всі тести R охоплюють всі можливі пари рівнів тест-факторів (що відносяться до різних параметрів). Іншими словами, для кожної пари рівнів факторів li,p та lj,q, де $1 \leq p \leq Li$, $1 \leq q \leq Lj$, та $i \neq j$ – існує принаймні один тест R, який містить li,p та lj , Q.

Для аналізу предметної галузі розробки тестів застосовується онтологічний підхід [35]. Під онтологією розуміється впорядкована трійка виду

$$O = (C, R, F), \tag{2.2}$$

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

R — кінцева безліч відносин між концептами заданої предметної галузі;

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

Тестування слід здійснювати з погляду користувача, 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).

У зв'язку з тим, що сучасному класу КС притаманний модульний принцип побудови, тобто. їх інтеграція, виділимо два підходи до комбінування модулів [38]: покрокове та монолітне тестування.

Перший підхід — монолітний метод, або метод «великого удару», застосовуваний при тестуванні та складанні програми.

Другий підхід — покроковий метод тестування або складання, що передбачає, що модулі тестуються не ізольовано один від одного, а підключаються по черзі для виконання тесту до набору вже раніше відтестованих модулів доти, доки до набору відтестованих модулів не буде підключений останній модуль.

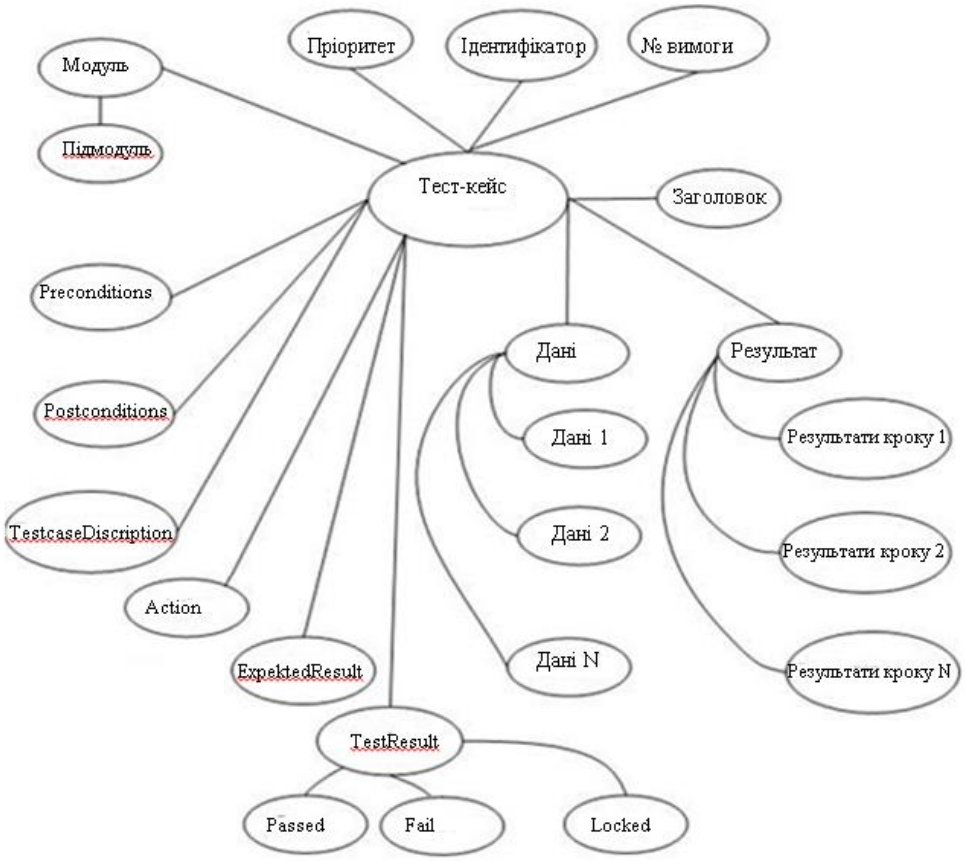


Рисунок 2.4 — Онтологічна модель тестового випадку

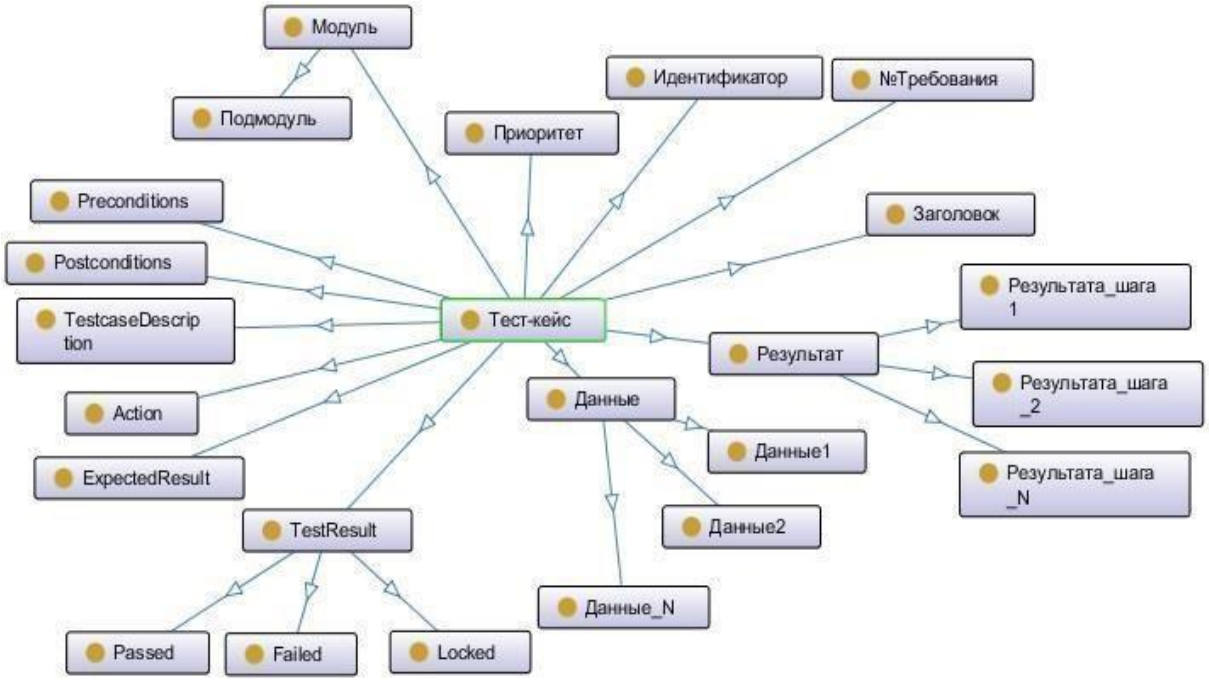


Рисунок 2.5 — Онтологічна модель тестового випадку в Protege

Аналіз виділених методів модульного тестування КС показав:

— монолітне тестування потребує великих витрат праці;

— при монолітному тестуванні менша витрата машинного часу;

— монолітний метод надає великі можливості для паралельної організації роботи на початковій фазі тестування (тестування всіх модулів одночасно);

— при покроковому тестуванні раніше виявляються помилки між модулями, оскільки раніше починається складання програми;

— при покроковому тестуванні легше налагодження програм;

— результати покрокового тестування більш досконалі та дають точніший аналіз у визначенні дефектів КС.

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

Інтеграційне тестування спрямоване на перевірку взаємодії між частинами (модулями) програмного забезпечення КС. Переконавшись у перевагах покрокового тестування перед монолітним, досліджуємо дві можливі стратегії тестування: низхідне та висхідне [6, 8, 13, 36].

Існують два основні підходи до проведення інтеграції: висхідна інтеграція (рис. 2.6); низхідна інтеграція (рис. 2.7).

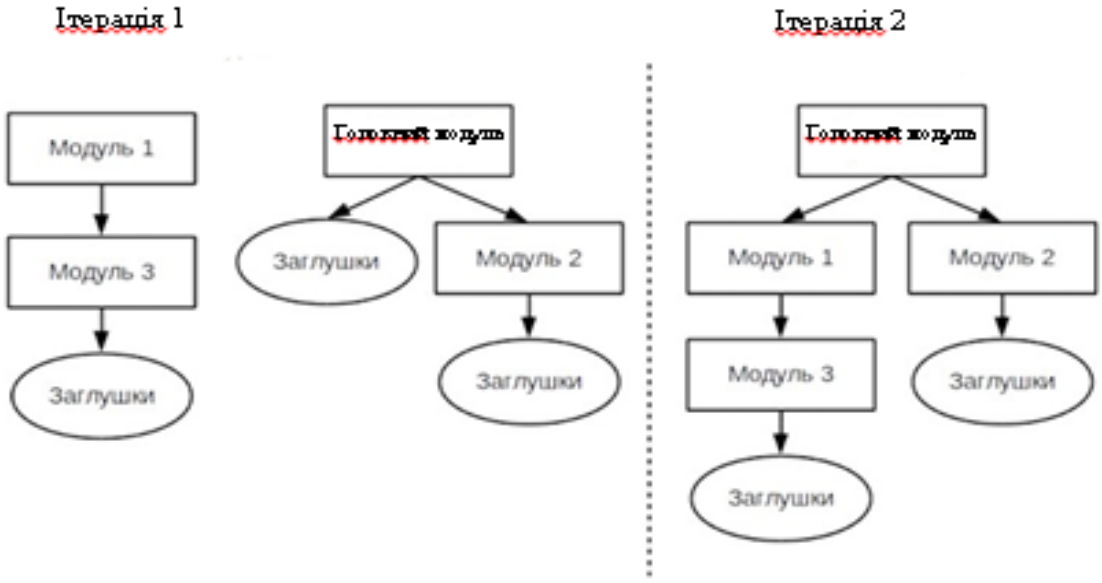


Рисунок 2.6 – Схема висхідної інтеграції

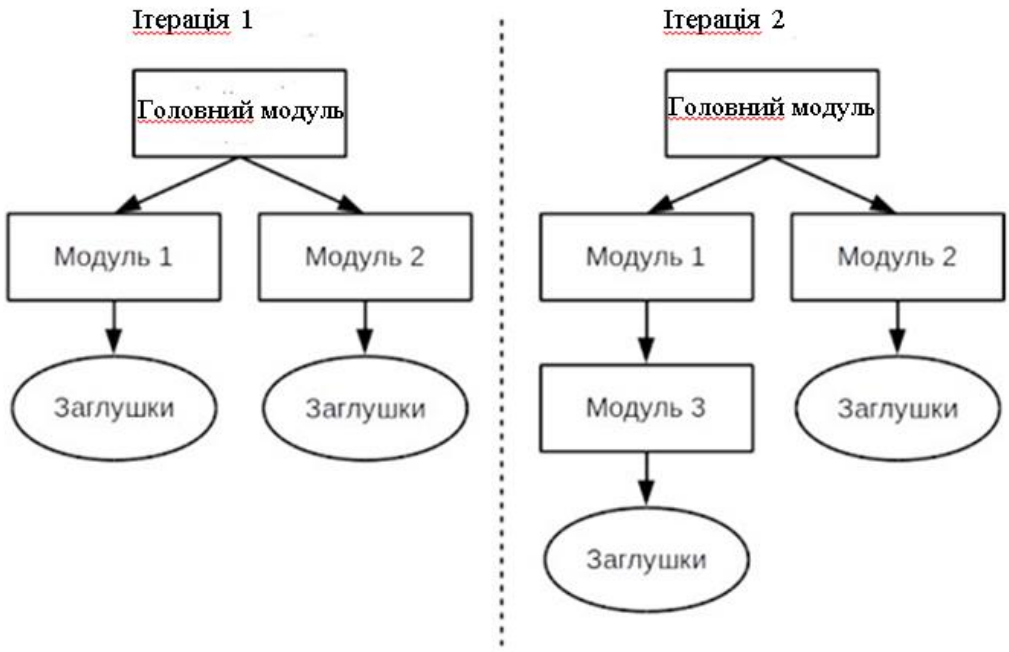


Рисунок 2.7 – Схема низхідної інтеграції

Переваги та недоліки представлених підходів відважені в таблиці 2.1.

Щоб процес тестування КС мав виправдану з економічного погляду трудомісткість, необхідно заздалегідь виробити ряд стратегій. Найбільш поширені:

— тестування методом «чорної» скриньки — тестування, кероване даними (data-driven testing) або тестування, кероване входом та виходом (input/output-driven testing);

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

Таблиця 2.1 — Порівняння способів інтеграції

	Висхідне	Східне
Переваги	1. Можливість ранньої перевірки коректності низькорівневої поведінки. 2. Не потрібно написання заглушок Просто визначити вимоги до входів/виходів модулів.	1.Можливість ранньої перевірки коректності високорівневої поведінки 2.Модулі можуть додаватися по одному, незалежно один від одного. 3. Не потрібна розробка безлічі драйверів. 4. Можна розробляти систему як у глибину, так і завширшки
Недоліки	1. Відкладена перевірка високорівневої поведінки 2. Потрібна розробка драйверів. 3. При заміні драйвера на модуль високого рівня може статися «міні Великий Вибух» числа знайдених помилок	1. Відкладена перевірка низькорівневої поведінки. 2. Потрібна розробка «заглушок». 3. Вкрай складно коректно сформулювати вимоги до входів/виходів часткової системи

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

Таблиця 2.2 — Порівняльний аналіз методів комплексного тестування

Тестування чорної скриньки	Тестування білої скриньки
1	2
Тестування чорної скриньки — це метод тестування програмного забезпечення,який використовується для тестування програмного забезпечення, не знаючи внутрішньої структури коду чи програми.	Тестування білої скриньки — це метод тестування програмного забезпечення, при якому внутрішня структура відома тестувальнику, який збирається тестувати програмне забезпечення.
Проводиться тестерами.	Проводиться розробниками.
Знання впровадження не потрібні для проведення тестування чорної скриньки.	Знання впровадження потрібно для проведення тестування білої скриньки.
Знання з програмування не потрібні.	Знання з програмування необхідні

Продовження табл. 2.2 — Порівняльний аналіз методів комплексного тестування

1	2
Тестування застосовується на вищих рівнях тестування, таких як системне тестування, приймальне тестування.	Тестування застосовується на нижчих рівнях тестування, таких як модульне тестування, інтеграційне тестування.
Означає функціональне тестування або зовнішнє тестування.	Означає структурні випробування або внутрішні випробування.
Концентрується на функціональності тестованої системи.	Концентрується на тестуванні програмного коду системи, що тестується, такий як структура коду, гілки, умови, цикли тощо.
Основна мета — перевірити, які функції виконує система, що тестується.	Основна мета — перевірити, як працює система.
Може бути почато на підставі документів із технічними вимогами.	Може бути почато на підставі документів робітничого проекту.
Функціональне тестування,тестування поведінки, тестування закритого боксу виконується у межах тестування «чорної скриньки», тому знання у сфері програмування не потрібні.	Структурне тестування, логічне тестування, тестування контурів, циклічне тестування, тестування покриття коду, тестування Open Box виконується в рамках тестування «білої скриньки», тому необхідно обов'язково знати про знання програмування.

Для комплексного тестування КС також застосовуються нижче приведені методи [37].

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 бета-тестування.

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, щоб знизити кількість дефектів. Іноді версія передається кільком командам, щоб отримати зворотний зв'язок від якнайбільшої кількості майбутніх користувачів.

Ключові переваги:

— отримання відгуків та побажань від потенційних користувачів КС;

— 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 КС.

2.3 Тестування комп’ютерної системи та її продуктивності

Сучасні КС 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 КС, а також керувати нею в умовах планового, 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 КС.

2.3.1 Тестування продуктивності

Тестування продуктивності (Performance Testing) — це тестування, яке виконується визначення того, як компоненти системи працюють у конкретній ситуації [32]. Основна мета тестування продуктивності включає встановлення еталонної поведінки системи. Тестування продуктивності не спрямовано 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 відповідає програмне забезпечення ІВ 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 [27]:

— визначте своє середовище тестування — ознайомтеся з деталями апаратного, програмного забезпечення та мережевих конфігурацій, використаних під час тестування;

— визначте критерії прийнятності продуктивності — сформулюйте цілі та обмеження пропускної спроможності, часу відгуку та розподілу ресурсів, критерії успіху проекту поза цими цілями та обмеженнями;

— плануйте та проектуйте тести продуктивності — визначте, як використання може змінюватись серед кінцевих користувачів, та визначте ключові сценарії для тестування у всіх можливих випадках використання;

— налаштування середовища тестування — підготуйте середовище тестування перед виконанням, організуйте інструменти та інші ресурси;

— реалізувати тестовий дизайн — створіть тести продуктивності відповідно до тестового дизайну;

— запустити тести — виконайте та оцініть результати тести;

—аналізуйте, налаштовуйте та повторно тестуйте — об'єднуйте, аналізуйте та ділитесь результатами випробувань; виконайте точне налаштування та перевірте знову, щоб побачити, чи є покращення або 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 в кошик та здійснення покупки.

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

Сценарій create_car включає три клієнтські запити, які читають список всіх автомобілів, створюють автомобіль та підключаються до створеного ресурсу. Сценарії настроюють кілька віртуальних користувачів. Кількість користувачів починається з 10 і збільшується до 20 протягом 10 секунд. Тестування продуктивності 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 із 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;

```
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)

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

Рисунок 2.8 — Фрагмент опису обробки клієнтських запитів

— 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.

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

2.3.2 Навантажувальне тестування

Навантажувальне тестування (load testing) — даний тип тестування дозволяє оцінити поведінку системи при зростаючому навантаженні. Метою тесту навантаження є визначення максимального навантаження, яке може витримати система [28].

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 вибраних операцій припевні 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, 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.

Інший приклад навантажувальноготесту показано на рис. 2.10 де зображено тест на піки при раптовому збільшенні кількості користувачів до 7000.

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

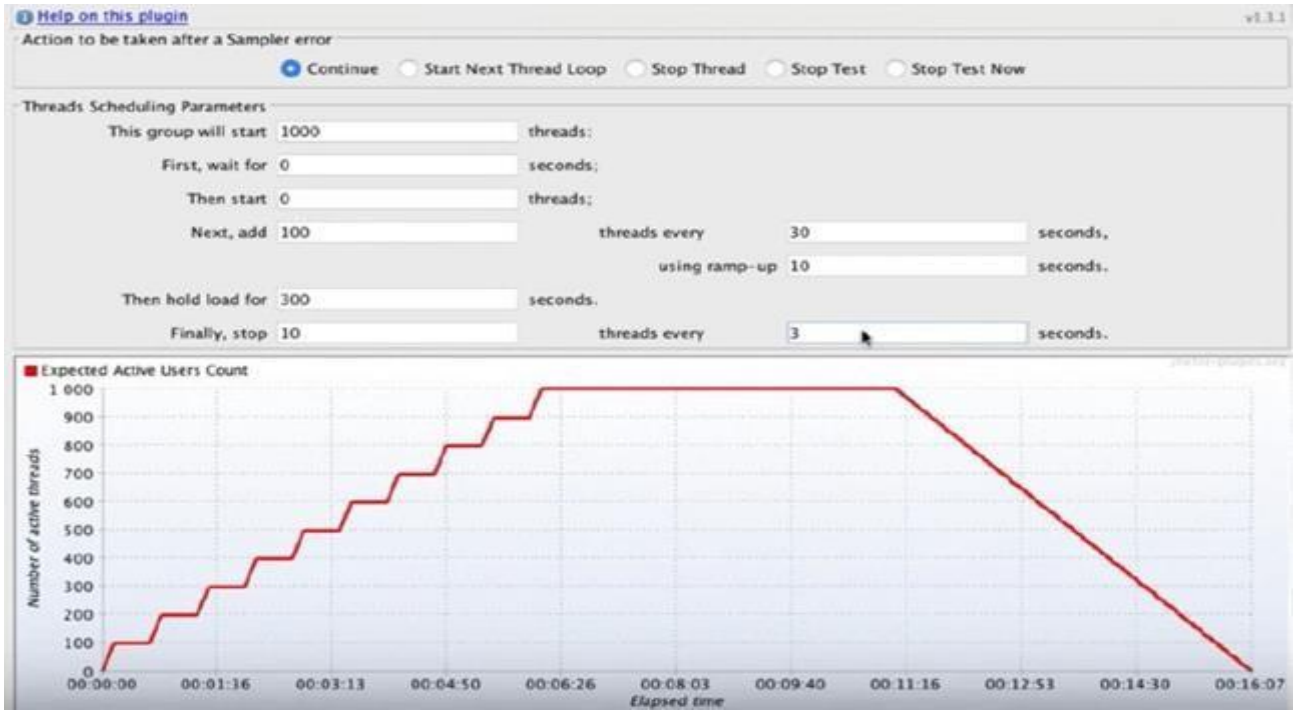


Рисунок 2.9 – Приклад виконання тесту навантаження

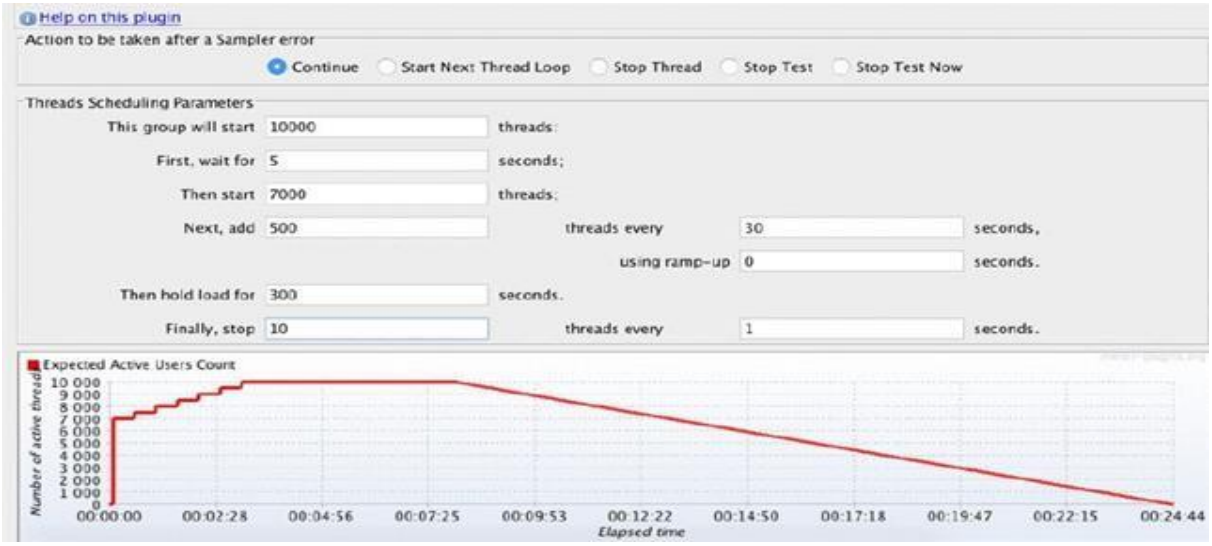


Рисунок 2.10 – Приклад виконання тесту навантаження

2.3.3 Стрес-тестування

При стрес-тестуванні виконуються різні дії з перевантаження існуючих ресурсів надмірними завданнями спробі зламати систему [44]. 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. 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

1	2	3	4
	Тестування продуктивності	Тестування навантаження	Стрес-тестування
Домен	Суперсет навантажувального та стрес-тестування	Підмножина тестування продуктивності.	Підмножина тестування продуктивності.
Об `єм	Дуже широка сфера застосування. Включає навантажувальне тестування, стрес-тестування, тестування ємності, об'ємне тестування, тестування намасштабованість і тестування надійності та ін.	Вужче охоплення при порівнянні з тестуванням продуктивності. Включає об'ємні випробування та випробування на витривалість.	Вужче охоплення при порівнянні з тестування продуктивності. Включає тестування напитування і тестування шпильок.
Основна ціль	Щоб встановити орієнтир та стандарти для програми.	Щоб визначити верхню межу системи, та подивіться, як система обробляє томи із великим навантаженням.	Визначити, як система веде себе при інтенсивних навантаженнях та як відновлюється після збою, щоб підготувати ваш додаток до несподіваного стрибка трафіку.
Межа навантаження	Обидва — нижче та вище порога перерви.	До порога розриву	Вище порога розриву

Продовження табл 2.3 — Порівняльний аналіз методів тестування продуктивності

	1	2	3
Атрибути вивчені	Використання ресурсів, надійність, масштабованість, використання ресурсів,час відгуку, пропускна здатність, швидкість і т.д.	Пікова продуктивність, пропускна здатність сервера, час відгуку різних рівнях навантаження (нижче порога перерви), адекватність робочої середовища, кількість користувальницьких додатків, вимоги до балансування навантаження і т.д.	Стабільність за межами пропускної здібності,час відгуку (вище порога розриву) тощо.
Проблеми, виявлені за допомогою цього типу тестування	Всі помилки продуктивності, включаючи роздування в час виконання, можливості оптимізації, проблеми, пов'язані зі швидкістю, затримкою, пропускною здатністю і т.д.	Проблеми з балансуванням навантаження, проблеми з пропускною здатністю, проблеми з пропускною здатністю системи, поганий час відгуку, проблеми з пропускною здатністю і т.д.	Пробілів безпеки перевантаженням, проблеми з пошкодженням даних у ситуації перевантаження, повільність, витікання пам'яті і т.д.

В 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:

— найбільша інтенсивність виконання операцій з необхідною якістю обслуговування (SLA);

— пікова продуктивність системи, коли він відбувається погіршення показників якості обслуговування операцій (час виконання, відмови).

2.4 Методи та сценарій приймального тестування

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 [41].

Ключові переваги:

— дозволяє виявити системні порушення;

— дозволяє виявити дефекти, пов'язані із зручністю та простотою використання.

— залучення досвідчених компетентних фахівців дозволяє грамотно, якісно та у задані терміни провести процес приймання тестування.

Основні етапи приймального тестування:

— підготовка — включає розробку програми та методики випробувань та підготовку приймальних тестів;

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

— 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 повинен утримувати чітко сформульовані розділи, представлені рисунку 2.11.

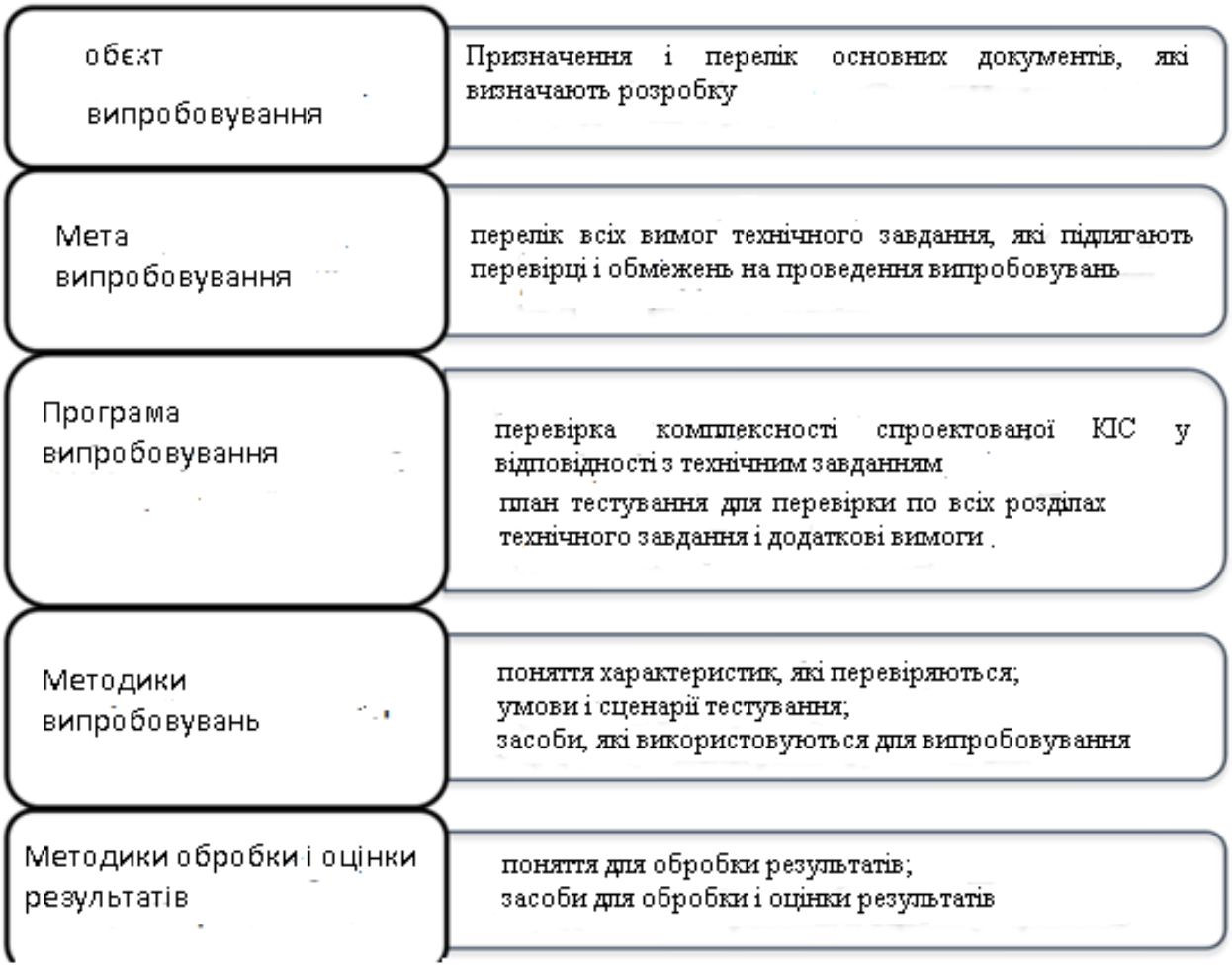


Рисунок 2.11 — Сценарій приймального тестування

Найбільш повним та різнобічним приймальним випробуванням має піддаватися перша базова версія КС. Комплексне тестування програмного забезпечення КС гарантує перевірку виконання всіх вимог технічного завдання.

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 КС. Для цього необхідно проводити випробування у два послідовні етапи – Альфа- та Бета-тестування [12, 26]. Обсяг тестів та тривалість обох етапів тестування залежатимуть від складності комплексу програмного забезпечення КС та інтенсивності потоку змін.

3 АПРОБАЦІЯ ЗАСОБІВ ТЕСТУВАННЯ ВІДМОВОСТІЙКОСТІ КОМПЮТЕРНОЇ СИСТЕМИ

3.1 Навантажувальне тестування, засноване на моделях

Розглянемо підхід, заснований на моделях, стосовно навантажувального тестування КС, в рамках якого здійснюється тестування прикладного програмного забезпечення (ПЗ) в IT-середовищі функціонування, що включає СУБД, бази даних, системне ПЗ, розгорнуті на обладнанні комплексу технічних засобів КС [29, 30 , 31].

При навантажувальному тестуванні КС у процесі експлуатації головною метою не перевірка правильності функціонування прикладних програм (передбачається, що функціональне тестування ПЗ проведено), а оцінка експлуатаційних характеристик інформаційної системи, у складі якої функціонує прикладне ПЗ. Зазвичай це так звані «Показники призначення», або нефункціональні вимоги, тобто пропускна спроможність/продуктивність КС, реактивність при вирішенні функціональних завдань та низка інших.

При формалізації предметної області (для класу систем) формуються вихідні дані [30]:

— інформація про обраний вид навантажувального тестування (оцінне, аналітичне, настроювальне, регресійне);

— інформація про характеристики, що вимірюються і показники продуктивності;

— інформація про структуру системи з погляду варіантів подачі навантаження та способів вимірювань;

— інформація про завантаження в структурованому вигляді.

Представимо такі моделі, за допомогою яких при постановці завдання навантажувального експерименту здійснюється вибір необхідних характеристик, показників та вимірюваних величин, що адекватно характеризують процес функціонування КС, що тестується (рис. 3.1):

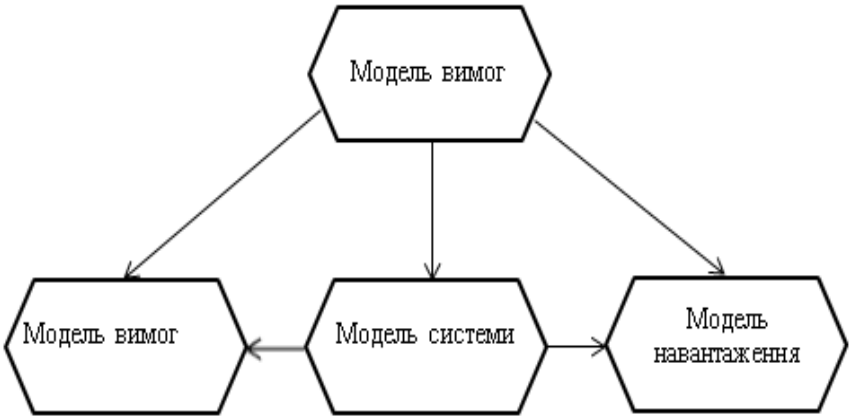


Рисунок 3.1 — Взаємозв'язок моделей

Модель вимог – характеризує тип системи, що тестується, склад дисфункційних вимог (бізнес-правила та технічні вимоги) тестованої інформаційної системи [31].

Модель вимог може бути подана у такому вигляді:

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

де R — безліч вимог до системи;

B — безліч бізнес-правил, пов'язаних з технологічними процесами, корпоративними регламентами, політиками, стандартами, законодавчими актами, внутрішньо корпоративними ініціативами, обліковими практиками, алгоритмами обчислень тощо;

T — безліч технічних вимог, що встановлюють технічні властивості, якими повинна мати система, наприклад, характеристики продуктивності, надійності та доступності.

Модель вимог є вербальний опис нефункціональних вимог, згрупованих за типами і об'єктами, до яких вони застосовні. Ця модель містить також критерії оцінки одержуваних у результаті навантажувального тестування характеристик та розрахункових показників.

Модель системи — визначає структуру системи як мережу систем масового обслуговування (включаючи склад елементів типу «ресурс») [31].

Модель системи може бути представлена у такому вигляді:

$$\{U(p)\}, \{S\}, K_S, K_U \tag{3.2}$$

де {U(p)} — безліч пристроїв об'єкта тестування з характеристиками продуктивності p;

{S} — безліч програмних комплексів (компонент, сервісів);

K_S — матриця зв'язків між програмними комплексами, рядками якої є джерела, стовпцями — приймачі, а осередках вказується наявність зв'язку з-поміж них;

K_U — матриця зв'язків програмних комплексів із пристроями, що характеризує кількість виділених пристроєм ресурсів для програмного комплексу.

Рядками цієї матриці є програмні комплекси, стовпцями — обчислювальні комплекси. Елементами матриці є вектори виділених ресурсів. Матриці K_S та K_U для представленої схеми виглядають наступним чином:

$$K_S = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} K_U = \begin{pmatrix} p_1 & 0 & 0 & 0 \\ 0 & p_2 & 0 & 0 \\ 0 & 0 & p_3 & 0 \\ 0 & 0 & 0 & p_4 \\ 0 & 0 & 0 & p_4 \end{pmatrix}$$

де p_i — вектор характеристик продуктивності, що характеризує кількість виділених на обчислювальному комплексі ресурсів для заданого програмного комплексу.

Модель системи являє собою опис об'єкта тестування, до якого входять програмні та технічні засоби, їх зв'язки та ресурси, що виділяються (пам'ять,

процесор і т.п.). На рисунку 3.2 представлений приклад моделі системи у вигляді графічної схеми взаємодії програмних та обчислювальних комплексів.

Модель навантаження — є опис кількості та типів вимог обслуговування до системи, закон розподілу вимог обслуговування в часі експерименту, правила надходження вимог обслуговування до системи, точки входу вимог обслуговування до системи (логічний рівень) [29].

Модель навантаження може бути представлена у такому вигляді:

$$L= \{F,M,I\} \quad (3.3)$$

де F — безліч функцій, що характеризують розподіл навантаження, що вводиться в систему;

M — багатовимірна матриця, розмірностями якої можуть бути види навантаження: джерела потоків навантаження, найменування та типи потоків, а елементами — їх кількісні характеристики;

I — безліч інтерфейсів для введення навантаження (як посилання на модель системи).

Потік навантаження структурується за його динамічними (F) та статичними властивостями.

Модель навантаження є описом тестових даних і правил їх надходження в систему. Фактично за моделлю навантаження формуються вихідні дані для генерації навантаження інструментальними засобами.

Модель вимірювань — визначає склад характеристик, показників і величин, що збираються, інтерфейс введення вимог до системи, метод їх збору та алгоритми перетворення, а також критерії оцінки отриманих результатів [28].

Модель вимірювання призначена для уніфікації опису:

— способів отримання вимірюваних величин навантажувального тестування КС;

— важливих можливостей постановки процесу вимірів;

— типових способів оцінки показників та характеристик;

— загальних властивостей інструментальних засобів для аналізу можливостей їх використання для автоматизації вимірів.

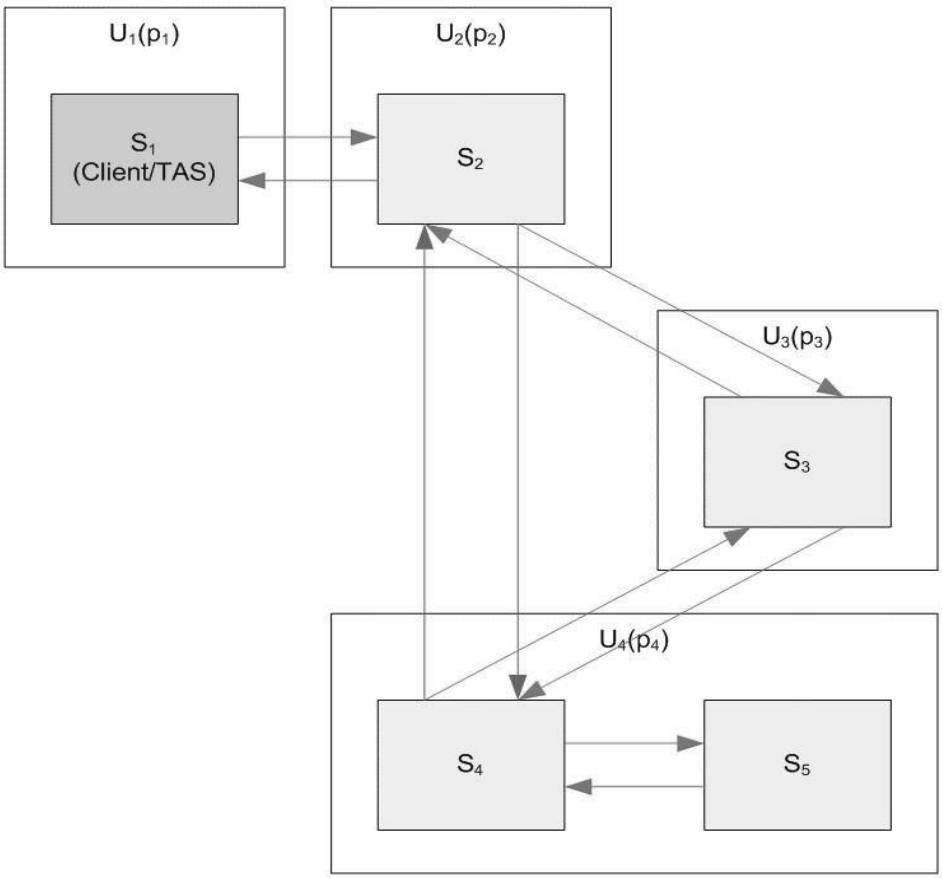


Рисунок 3.2 — Приклад моделі системи як графічної схеми

Модель вимірювання може бути представлена в наступному вигляді:

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

де {|U|} — перелік вимірюваних величин кожному за типом пристроїв КС чи її частини;

τ — періодичність та шпаруватість вимірювань; μ — безліч розрахункових показників та їх взаємозв'язок із вимірюваними величинами;

R — типові правила та алгоритми отримання розрахункових показників;

Ω — типові критерії оцінки отриманих результатів.

Для планування експерименту навантаження ключову роль грає вибір переліку вимірюваних характеристик продуктивності (табл. 3.1), так як на основі одержаних значень цих характеристик робляться висновки про результати експерименту.

Реактивність важлива для КС, експлуатаційними характеристиками КС можуть бути час відгуку (реакції) системи на запит користувача або час очікування розв'язання задачі (наприклад, може вирішуватись задача мінімізації часу очікування виконання бізнес-транзакції).

Таблиця 3.1 — Види показників продуктивності

Характеристики продуктивності	Розрахункові показники
Реактивність	Середній час відгуку
	Середній час очікування
	Середній час обслуговування
Продуктивність	Пропускна спроможність
	Вироблення
Використання	Утилізація ресурсу
	Відносна пропускна спроможність

Характеристики реактивності визначаються як час між пред'явленням системи вхідних даних та появою відповідних вихідних даних. Реактивність оцінюють або розраховують на основі наступних показників продуктивності:

— середній час відгуку;

— середній час обслуговування;

— середній час очікування.

Час відгуку розраховується як період між початком транзакції та завершенням виконання останнього етапу транзакції.

Час відгуку (Tr) зазвичай складається з часу обслуговування (час, за який проводиться обробка) (Ts) та часу очікування (час очікування ресурсу) (Tw): Tr = Ts + Tw.

Показники реактивності залежать від типу системи, структури її точок входу та виходу.

Для КС важлива їх інтегральна пропускна спроможність, що оцінюється як кількість бізнес-транзакцій, що обробляються системою в одиницю часу (продуктивність) [38].

При оцінці продуктивності зазвичай використовують безпосередньо виміряні або розрахункові значення наступних показників продуктивності:

— вироблення (V);

— пропускна спроможність (або абсолютна пропускна спроможність) ©.

Виробіток (V) визначається як кількість завершених транзакцій за певний період:

$$V = \frac{N_i}{T} \tag{3.5}$$

де 1, ..., n — порядковий номер періоду часу;

Ni — кількість завершених транзакцій;

T — період часу.

Пропускна спроможність (C) — максимально можлива кількість завершених транзакцій за одиницю часу:

$$C = \max(V1,...,Vn). \tag{3.6}$$

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 спроможність.

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

$$U = \frac{\sum_{i=1}^{n} t_{Si}}{T} \times 100\% \tag{3.7}$$

де n — кількість обслуговуваних транзакцій;

tSi — час обслуговування першої транзакції;

T — період обслуговування.

Характеристики використання ресурсів та їх кількість істотно залежать від використовуваного в системі обладнання та входять до його складу ресурсів: процесорів, оперативної пам'яті, зовнішніх носіїв, каналів введення-виведення тощо.

Технологія тестування навантаження, заснована на моделях (рис. 3.3), складається з наступних етапів [29]:

— 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 точки входу, задані для кожного типу навантаження в моделі системи;

— збирання даних — виконується автоматизований збирання значень вимірюваних характеристик;

— інтерпретація та аналіз результатів — засобами автоматизації 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.

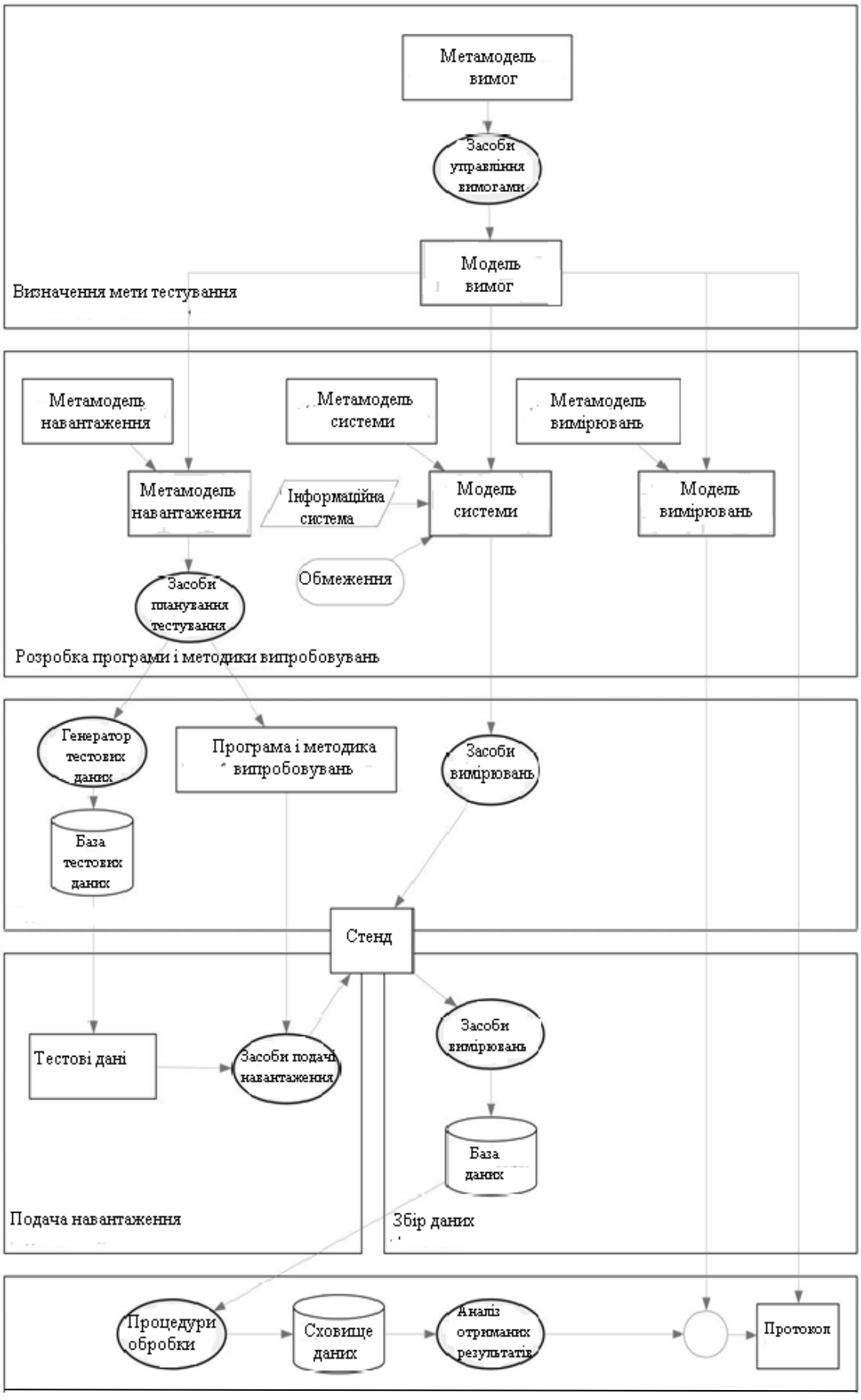


Рисунок 3.3 — Процес навантажувального тестування з використанням моделей

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 істотно (у кілька разів) знижує трудомісткість тесту навантаження і забезпечує можливість повторного використання підготовленого експерименту, наприклад, при контролі деградації інформаційної системи в її життєвому циклі.

3.2 Тестування навантаження з використанням різних інструментальних засобів

Розглянемо тестування навантаження для оцінки відмовостійкості простого HTTP-запиту GET з 20 потоків з 100 000 ітерацій.

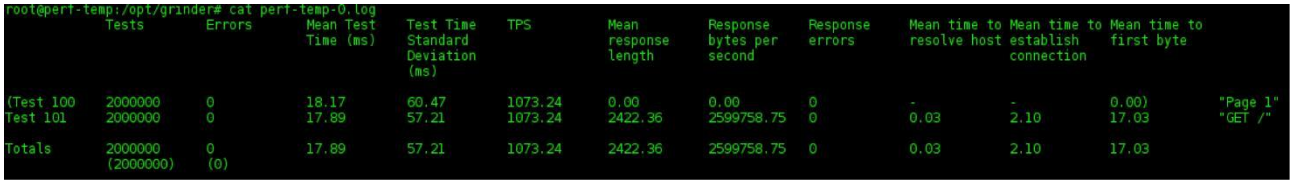
Сторона сервера (додаток, що тестується): Процесор: 4x Xeon L5520 @ 2,27 ГГц; RAM: 8 ГБ; ОС: Microsoft Windows Server 2008 R2 x64; Сервер програм: IIS 7.5.7600.16385.

Клієнтська сторона (генератор навантаження): Процесор: 4x Xeon L5520 @ 2,27 ГГц; RAM: 4 ГБ; ОС: Ubuntu Server 12.04 64-бітна.

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

Grinder — це безкоштовне середовище тесту навантаження на основі Java, доступне за ліцензією BSD в стилі open-source.

На рис. 3.4 представлений фрагмент проведення тесту навантаження, виконаного з використанням Grinder.



```
root@perf-temp:/opt/grinder# cat perf-temp-0.log
             Tests      Errors      Mean Test    Test Time     TPS         Mean        Response    Response    Mean time to  Mean time to  Mean time to
                                    Time (ms)    Standard                  response    bytes per   errors      resolve host  establish     first byte
                                                 Deviation                 length      second                  connection
                                                 (ms)

(Test 100    2000000     0           18.17        60.47        1073.24     0.00        0.00        0           -             -             0.00)          "Page 1"
Test 101     2000000     0           17.89        57.21        1073.24     2422.36     2599758.75  0           0.03          2.10          17.03          "GET /"

Totals       2000000     0           17.89        57.21        1073.24     2422.36     2599758.75  0           0.03          2.10          17.03
             (2000000)   (0)
```

Рисунок 3.4 — Навантажувальне тестування за допомогою Grinder

Gatling Project — це інструмент для тестування продуктивності з відкритим вихідним кодом, який в основному розроблений та підтримується Stephane Landelle. На рис. 3.5 наведено приклад звіту Gatling для сценарію завантаження.

Рисунок 3.5 — Приклад звіту тестування навантаження з Gatling

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 рисунках 3.7 наведено статистичний звіт навантажувального тестування, 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

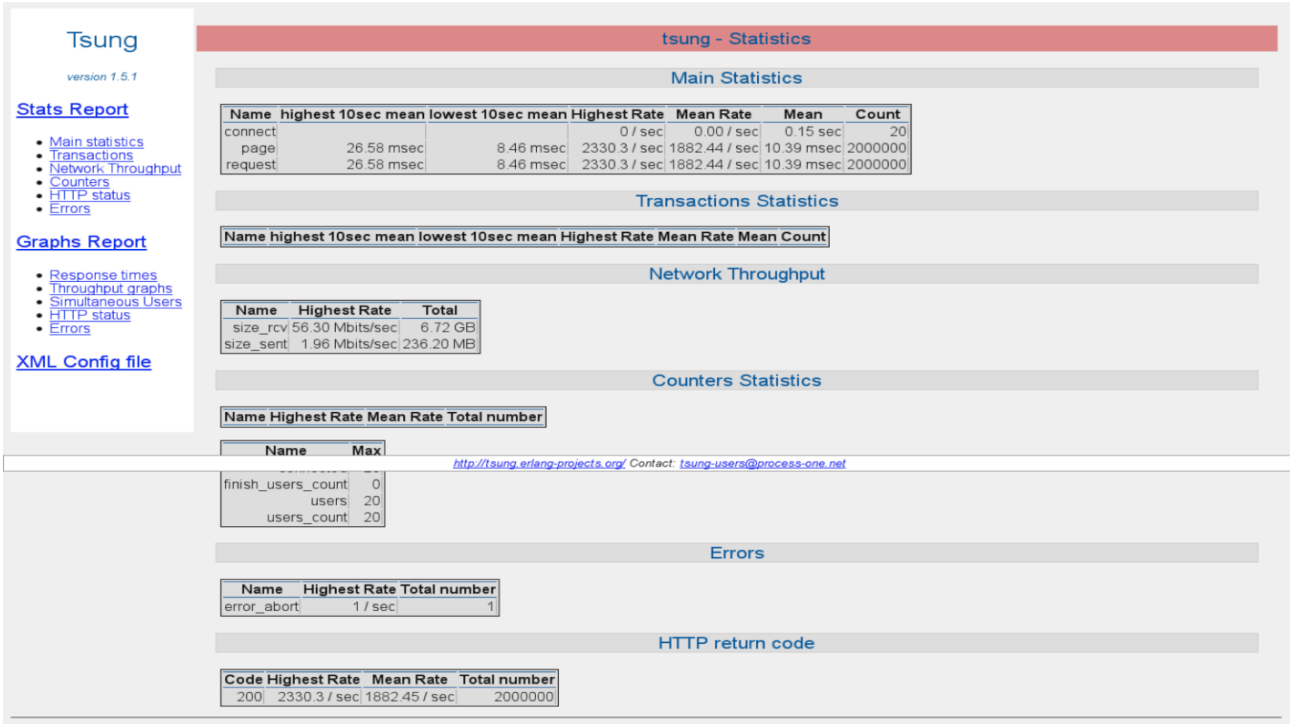


Рисунок 3.7 — Статистичний звіт навантажувального тестування

Рисунок 3.8 — Графічний звіт навантажувального тестування

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

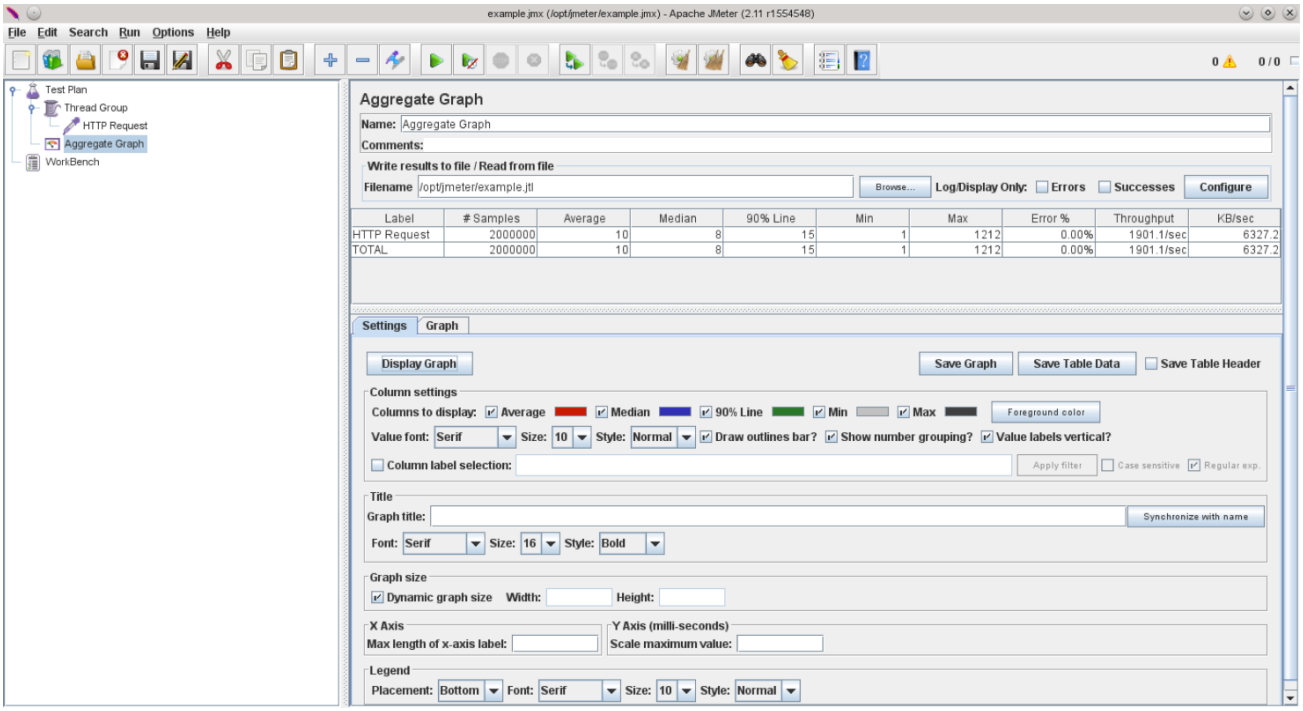


Рисунок 3.9 — Агрегований звіт сценарію навантаження JMeter

Locust заснованої на Python середовищі з відкритим вихідним кодом, що дозволяє писати скрипти продуктивності чистою мовою Python.

Приклад базового тестового сценарію з висновком представлений 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
[2022-11-18 14:46:38,458] Yuris-MBP-2.home/INFO/locust.main: Starting web monitor at *:8089
[2022-11-18 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 скрипту відображається докладна звітність (рис. 3.11).

Рисунок 3.11 — Звітність тестового сценарію Locust

Порівняємо результати навантажувального тесту цих інструментів зі наступними показниками:

— середній час відгуку (мс);

— середня пропускна спроможність (запитів/сек);

— загальний час виконання тесту (хвилин).

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

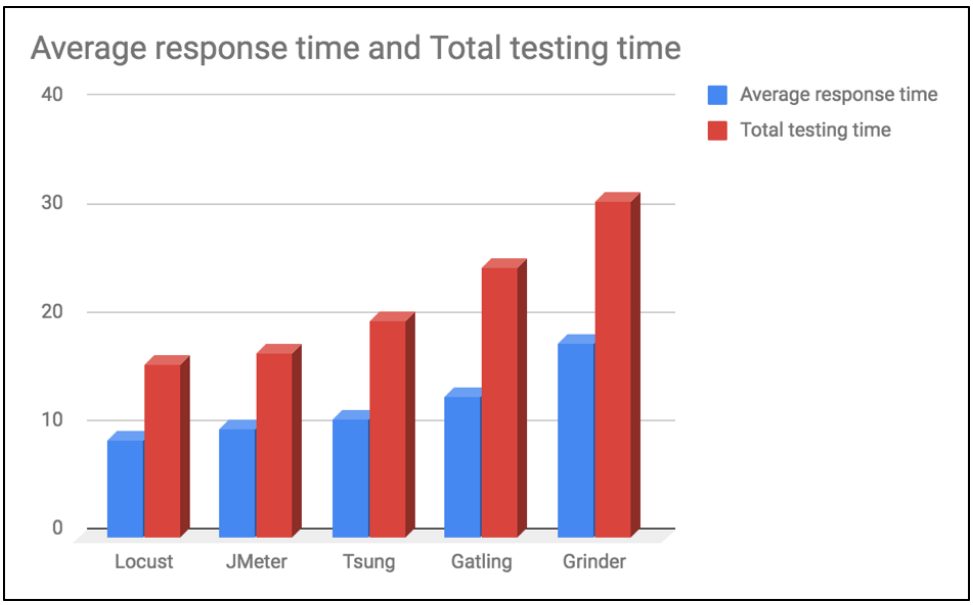


Рисунок 3.12 — Процес навантаження тестування з використанням моделей

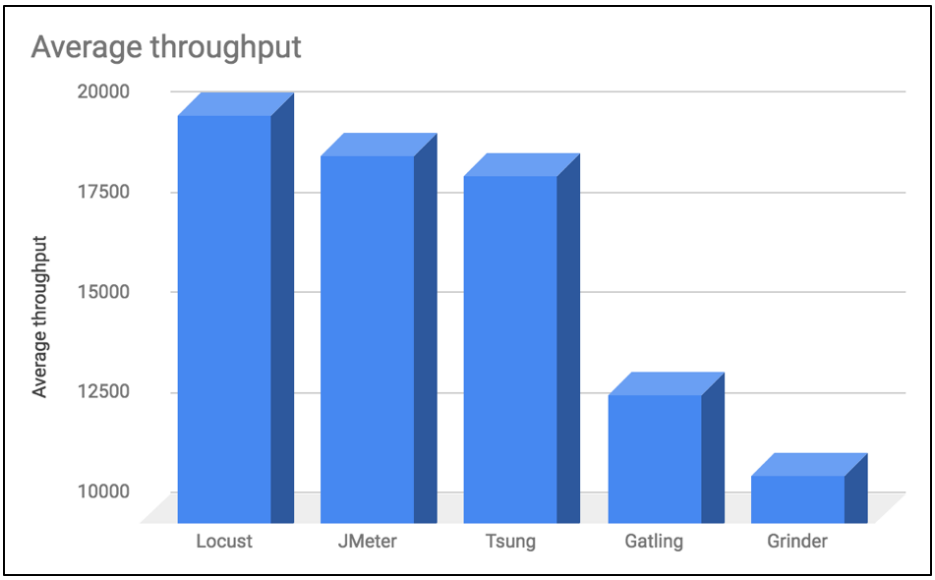


Рисунок 3.13 — Процес навантаження тестування з використанням моделей

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
1	2	3	4	5	6
Операційні системи	будь-який	будь-який	Linux/Unix	будь-який	будь-який
Графічний інтерфейс користувача	Тільки консоль	Тільки рекордер	ні	Повний	ні

Тест рекордер	TCP (включаючи HTTP)	HTTP	HTTP, Postgres	HTTP	ні
Тест Мова	Python, Clojure	Scala	XML	XML	Python
Мова Розширення	Python, Clojure	Scala	Erlang	Java,Beanshell, Javascript, Jexl	Python

Продовження табл.3.2 — Порівняльний аналіз інструментальних засобів навантажувального тестування

1	2	3	4	5	6
Завантажити звіти	Console	HTML	HTML	CSV,XML, вбудовані таблиці, графіки, плагіни	HTML
Протоколи	HTTP SOAP JDBC POP3 SMTP LDAP JMS	HTTP JDBC JMS	HTTP WebDAV Postgres MySQL XMPP WebSocket AMQP MQTT LDAP	HTTP FTP JDBC SOAP LDAP TCP JMS SMTP POP3 IMAP	HTTP
Моніторинг хоста	ні	ні	так	Так, з плагіном PerfMon	Ні
Обмеження	Знання Python потрібне для розробки та редагування тестів. Звіти дуже прості та короткі.	Обмежена підтримка протоколів. Потрібно знання мови DSL на базі Scala. Не 69абл.69остійко	Протестовано ї Підтримуєтьс я тільки в системах Linux.	Пов’язані звіти не легко інтерпретувати.	Знання Python потрібне 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 (рис. 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 обтяжуватися запитами (рис. 3.15). Запустимо тест Jmeter для оцінки використання ресурсів VPS (рис. 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

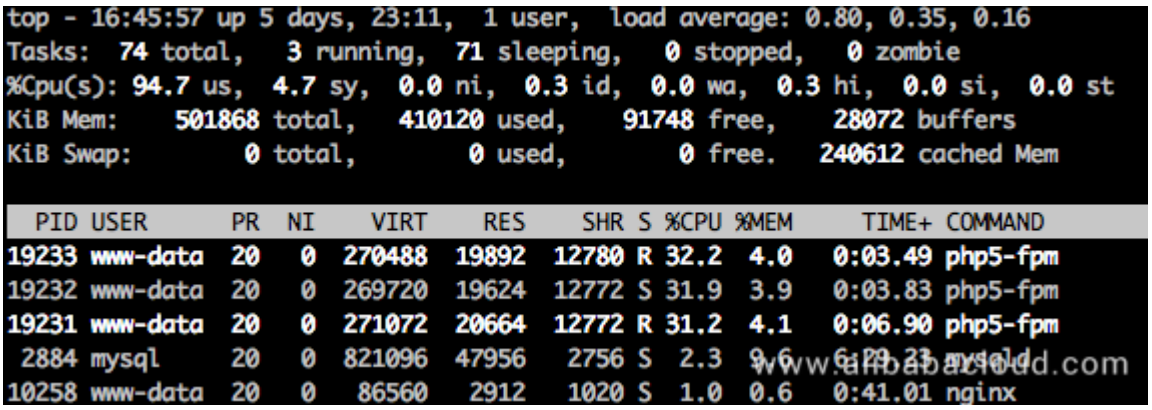


Рисунок 3.16 — Результати тестування зі збільшенням навантаження з використанням Jmeter

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 і як наслідок — поява збоїв, що свідчить про рівень порушення стійкості до відмов. Отже для перевірки продуктивності та 71абл.71о стійкості можна використовувати інструментальні засоби, наприклад, Jmeter має багато застосувань у контексті підвищення продуктивності та перевірки 71абл.71о стійкості та оптимізації серверів.

3.3 Тестування продуктивності та надійності бази даних

Розглянемо тестування надійності, продуктивності та, як наслідок, стійкості КС. Характеристики та список обладнання для проведення навантажувального тестування представлений у табл. 3.3.

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

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

— підтримка реляційної чи 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.

Таблиця 3.3 — Технічне забезпечення КС

Позиція	Позначення	Найменування та технічна характеристика
1	2	3
1	Маршрутизатор	Asus RT-AC68U
2	Сервер	Xeon E5-2630, 12 x 2.60 ГГц, процесорів 2, оперативна пам'ять – 8 ГБ, пам'ять – 2 x 500 ГБ
3	Зовнішнє сховище даних	BLOB у SharePoint Foundation
4	Джерело безперебійного живлення	Smart-UPS RT 3000VA RM 230V
5	Робочі станції	Dell Precision 7510 (4K IGZO) (Precision 7000 Серія) Процесор: Intel Xeon E3-1535M v5 2.9 GHz Графічний адаптер: NVIDIA Quadro M2000M, Ядро: 1038 - 1197 МГц МГц, 10.18.13.5445 - nVIDIA ForceWare 354.45, yes

		Оперативна пам'ять: 32768 Мбайт , DDR4-2133 Дисплей: 15.6 дюйм. 16:9,3840x2160пікс. 282 точок/дюйм, Sharp LQ156D1, IGZO IPS, глянсове покриття: немає Материнська плата: Intel Sunrise Point CM236 Зберігання даних: Samsung SSD SM951a 512GB M.2 PCIe 3.0 x4 NVMe (MZVKV512), 512 Гбайт
6	Роутер	TP-LINK TL-WA901ND
7	Лазерні багатофункціоналні пристрої	Canon i-SENSYS MF3010

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 з знятим показниками робляться висновки про кількість користувачів, що одночасно працюють.

Для результативного проведення процедури тестування КС необхідне певне технічне та програмне забезпечення завдання.

Визначено види застосовуваного тестування:

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

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.

Тест відмовостійкості виконується на рівні типового навантаження, який зазвичай встановлюється на рівні 70% від максимальної Lmax. Критеріями успішного проходження системою тесту є:

— 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 (рис. 3.17).

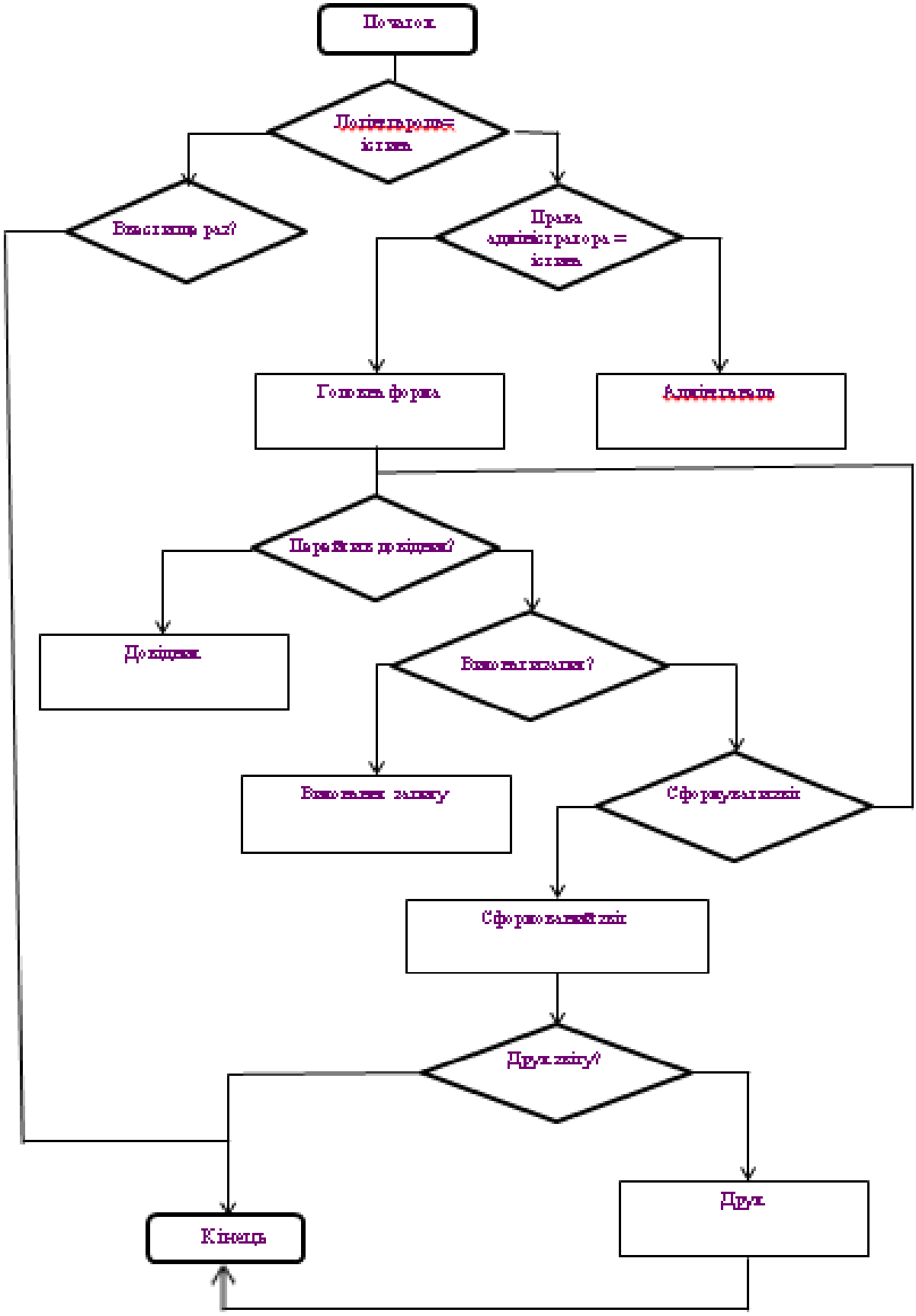


Рисунок 3.17 — Алгоритм процедури тестування за сценарієм 1

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

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, що виконуються віртуальними користувачами та зовнішніми системами через відповідні інтерфейси; інтенсивність операцій, що виконуються віртуальними користувачами та зовнішніми системами;

— 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 КС виконується на заздалегідь підготовленій базі з наступними параметрами:

— кількість співробітників, що одночасно працюють, до 50;

— кількість об'єктів обслуговування — 10000;

— кількість видів обслуговування кожного об'єкта — 4;

— кількість контрольованих показників кожного об'єкта 4-5;

— 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 КС здійснюється шляхом електронної пошти, локальною мережею або у формі документів. Достовірність введення даних буде реалізована за допомогою лічильного контролю. Формування звітів є механізм, заснований на використанні шаблонів звітів. Шаблони звітів є параметризованими звітами.

Аутентифікація користувачів у КС базується на механізмах, вбудованих у використовувану СУБД (Microsoft SQL Server або Sybase Adaptive Server). При цьому використовується концепція «наскрізної» автентифікації 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 отримати із співвідношення часу введення документа в базу 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.

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

де t_{prep} — сумарний час необхідної послідовності запитів до бази, що імітує заповнення реквізитів документа;

t_{pause} — пауза на хвилину;

t_{save}– час збереження документа.

Після проведення тестування необхідно оцінити відношення часу t_{save} до загального часу роботи користувача. Загальна кількість користувачів:

$$P = s/Rel \quad (3.9)$$

де s – максимально досягнута кількість одночасно працюючих сесій.

Розглянемо побудову марковської моделі для тестування навантаження КС 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, що означає, що у час запитів до БД не надходило, і модель перетворюється на 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 □t значення tj (j = 1…m, m – число значень на зазначеному інтервалі). Таким чином, представлений файл журналу можна формалізувати у вигляді безлічі трійок виду:

$$L = \{lj | lj = (tj, id, qi)\} \quad (3.10)$$

Далі розглянемо процес побудови марковської моделі за безліччю 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 переходить у стан si, при цьому ймовірність переходу у вказаний стан буде:

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

де n_{si} — кількість переходів у стан s_i;

m — загальна кількість розбиття часового інтервалу T.

В результаті обчислень формується четвірка виду:

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

де s_{n-1} — попередній стан моделі;

s_n — новий стан моделі;

q_i — запит, що викликав перехід у стан sn;

p — розрахункове значення ймовірності.

Безліч трійок є марковською модель навантажувального тестування КС під час роботи з базами даних. Далі зазначений процес повторюється, використовуючи як базовий стан sn. Процес побудови моделі завершується у тому випадку, якщо всі трійки у множині L відзначені.

Для реалізації алгоритму введемо операції MARK (l_i) – позначки певного елемента множини, операцію M(L), що повертає як результат потужність множини L, ЙМОВІРНІСТЬ(q_i) – результатом операції є ймовірність, розрахована за формулою (3.11), ++ – збільшення на 1. Тоді схема алгоритму побудови даної моделі виглядає як показано на рис. 3.18.

Слід зазначити, що марковську модель можна будувати як кожного клієнта БД, так всієї БД загалом.

Розглянемо далі безпосередньо процес тестування навантаження СУБД з використанням запитів КС. Нехай існує марківська модель (рис. 3.19).

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 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 та відмовостійкості.

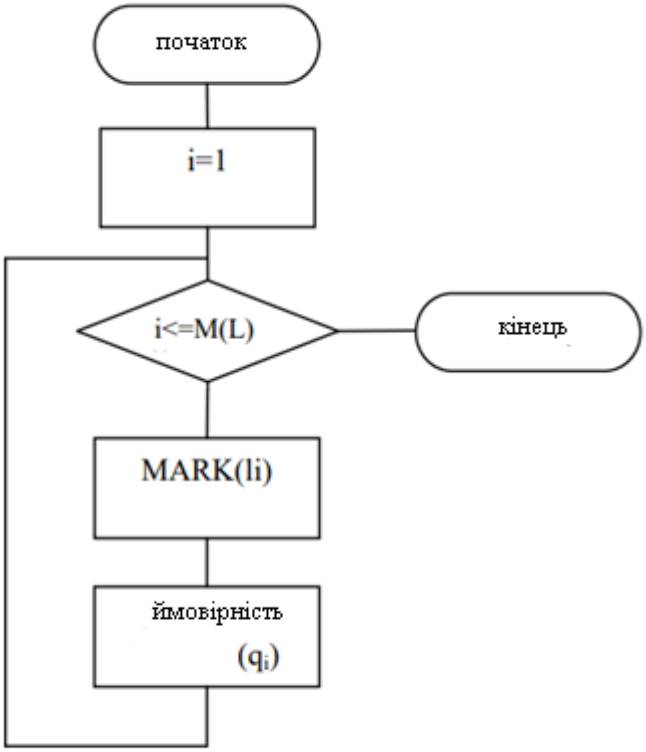


Рисунок 3.18 — Схема алгоритму побудови марковської моделі навантажувального тестування

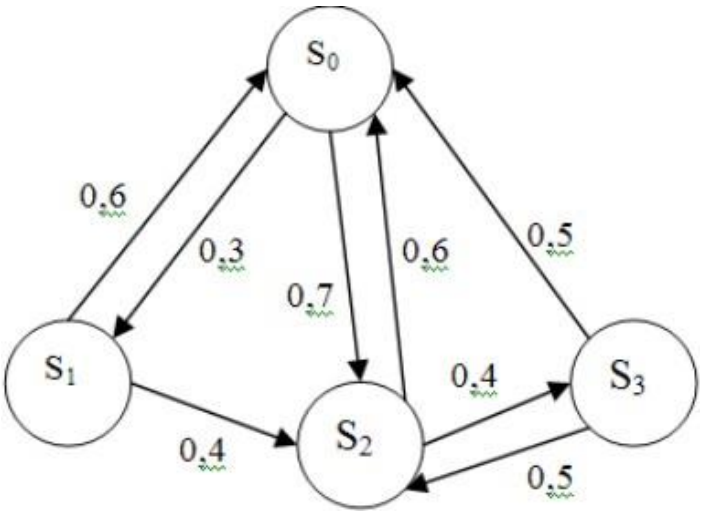


Рисунок 3.19 — Марківська модель

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 марковських моделях поведінки клієнтів СУБД, доцільно проводити тестування одним або декількома синтетичними тестами, структура запитів яких найбільше збігаються зі структурою запитів КС.

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 відмовостійкості та захищеності корпоративного оточення.

3.4 Навантажувальне тестування та формування критеріїв на доопрацювання

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

Рисунок 3.20 — Динаміка подачі 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 малюнках 3.21 і 3.22.

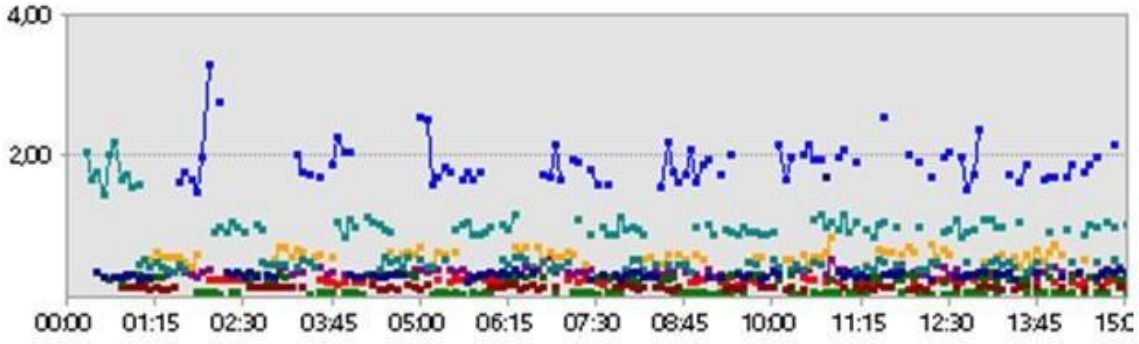


Рисунок 3.21 — Час відгуку Системи

Экземпляр	Цвет	Минимум	Максимум	Средн.
01 Авторизоваться в системе		0,86	2,23	1,09
02 Создать новый выпуск		0,24	1,75	0,34
03 Открыть вкладку Параметры выпусков		0,23	0,38	0,31
04 Выбрать выпуск Москва 44		0,12	0,20	0,16
05 Перейти в раздел Оценка эффективности		0,36	0,60	0,48
06 Создать новый сценарий выкупа		0,46	0,87	0,61
07 Ввести данные, сохранить и рассчитать		0,29	0,59	0,39
08 Открыть раздел Мониторинг и анализ		1,52	3,34	1,95
09 Открыть калькулятор и рассчитать цену		0,054	0,092	0,075
10 Выйти из системы		0,19	0,33	0,26

Рисунок 3.22 — Час відгуку Системи (легенда)

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

За результатами таблиці 3.4 видно, що час відгуку системи відповідає вимогам продуктивності при навантаженні 30 користувачів. Найменш продуктивними є транзакції: «13. Вибрати таблицю 1 та Сформувати», «15. 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 рисунках 3.23, 3.24.

Таблиця 3.4 — Статистика виконання сценарію

Транзакція	Мінімум (Сік)	Середнє (Сік)	Максимум (Сік)
01. Ввести невірний логін та пароль та перевірити аут	0,012	0,013	0,017
02. Увійти до системи під паролем	0,70	4,01	63,0
03. Відкрити розділ «Введення даних»	0,10	2,89	32,1
04. Відкрити паспорт	0,12	0,17	1,43
05. Відкрити паспорт.	0,11	0,15	0,71
06. Перейти до таблиці 1	0,054	0,37	25,4
07. Ввести 50 значень у таблицю та зберегти	0,080	1,19	32,4
08. Перейти до рівня ЗМ	0,087	0,54	8,94
09. Перейти до таблиці 4	0,12	0,36	19,6
10. Ввести 50 значень у таблицю та зберегти	0,12	0,43	27,9
11. Перейти до розділу Друковані форми	0,15	0,43	23,9
12. Вибрати програму та версію	0,16	0,21	0,28
13. Вибрати таблицю 1 та сформувати	5,84	6,33	7,29
14. Вибрати таблицю 4 та сформувати	1,05	1,28	1,72

15. Вибрати таблицю 2 та сформувати	9,67	14,2	43,9
16. Вийти із системи	0,028	0,057	0,13
17. Увійти до системи під admin	1,07	3,6	33,7
18. Відкрити вкладку «Моніторинг»	0,15	14,1	32,4
19. Відкрити розділ «Зведений звіт» про виконання	0,098	7,13	30,7
20. Вибрати 3 квартал 2022 року	2,05	3,81	20,7
21.Сформувати пояснювальну записку	0,12	0,14	0,22
22. Відкрити розділ «Зведений звіт»	2,07	2,28	2,61
23. Вийти із системи	0,033	0,043	0,070

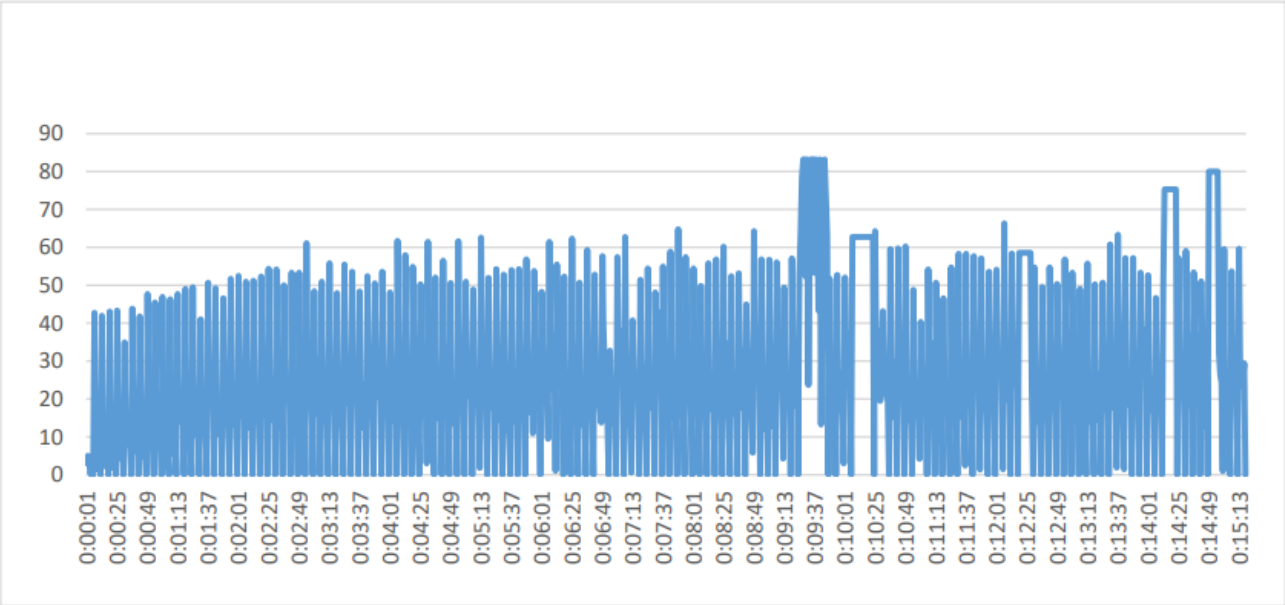


Рисунок 3.23 — Споживання ресурсів CPU сервера додатків

Рисунок 3.24 — Споживання пам'яті сервера програм

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

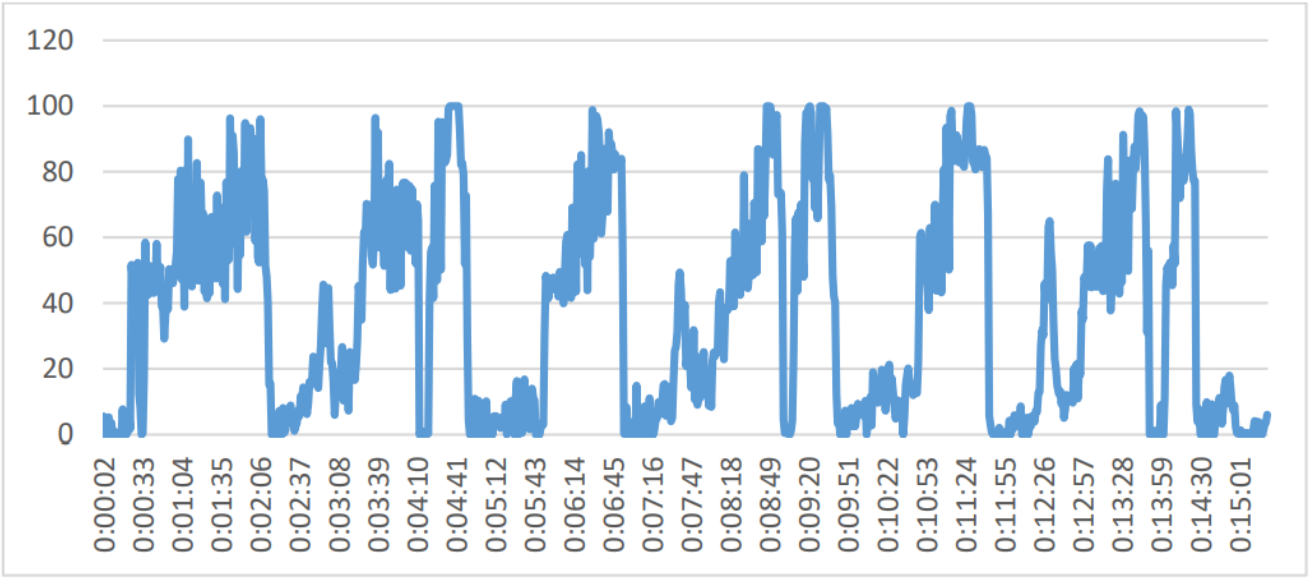


Рисунок 3.25 — Споживання ресурсів CPU сервера БД

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 і додатків перебуває в середньому рівні 10,4-11,3 гігабайт і 56,2-56,8 гігабайт.

В результаті проведеного тесту навантаження можна зробити наступні висновки:

— CPU сервера БД, 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 рекомендація розробникам системи слід звернути увагу на оптимізацію найменш продуктивних транзакцій «13. Вибрати таблицю 1 та Сформувати», «15. Вибрати таблицю 2 та Сформувати», Відкрити вкладку «Моніторинг».

3.5 Інтеграційне тестування

Проведено інтеграційне тестування з використанням платформи Citrus Framework.

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

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

Щоб змоделювати один із цих сценаріїв, був використаний Citrus Framework. Для перевірки його роботи була змодельована структура каталогів (рис. 3.28):

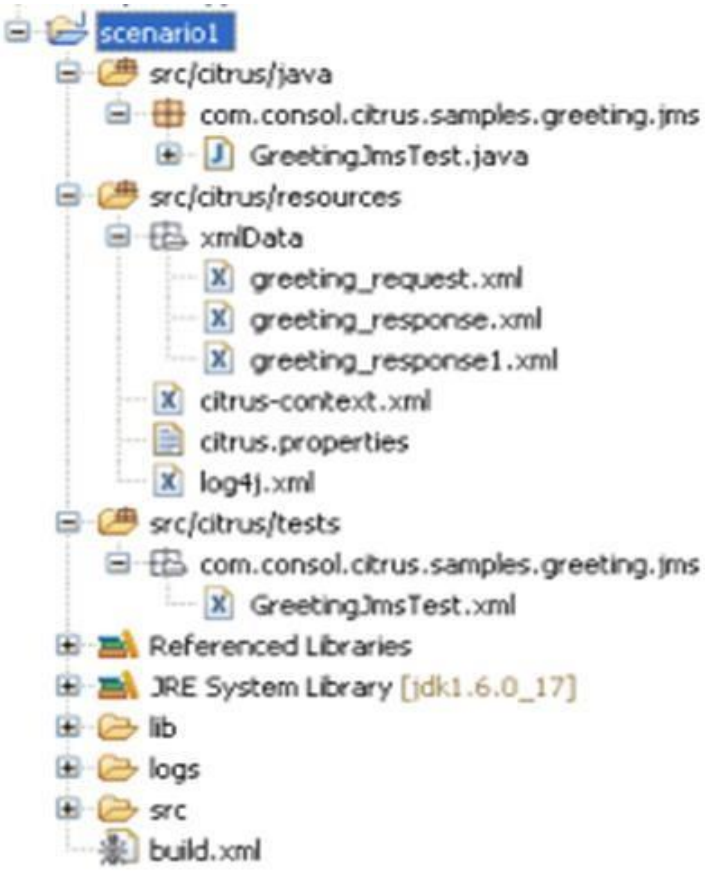


Рисунок 3.27 — Структура каталогів

— 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 зберігатиме логи (фрагменти корисного навантаження);

— build.xml — скрипт складання ant.

На рис. 3.28 наведено приклад використовуваного build.xml.

ПЗ КС 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.

```
<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

Стандартна конфігурація описує використання:

<citrus:jms-message-sender id="getOrdersRequestSender" destination-name="testJMSServer/citrus_queue_in"/>

Потрібно налаштувати src / citrus / tests / com / consol / citrus / samples /reeting/jms/GreetingJmsTest.xml (рис. 3.30).

```
<?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 такі команди:

```
<message>
    <resource file="classpath:xmlData/greeting_request.xml" />
</message>
```

або ж

```
<message>
    <data>
        <![CDATA[
            <tns:GreetingRequestMessage xmlns:tns="http://www.citrusframework.org/samp
                <tns:CorrelationId>${correlationId}</tns:CorrelationId>
                <tns:Operation>sayHello</tns:Operation>
                <tns:User>${user}</tns:User>
                <tns:Text>Hello Citrus!</tns:Text>
            </tns:GreetingRequestMessage>
        ]]>
    </data>
</message>
```

Таким чином, була проведена перевірка повідомлень XML та отримуємо результат, наведений на рисунку 3.31.

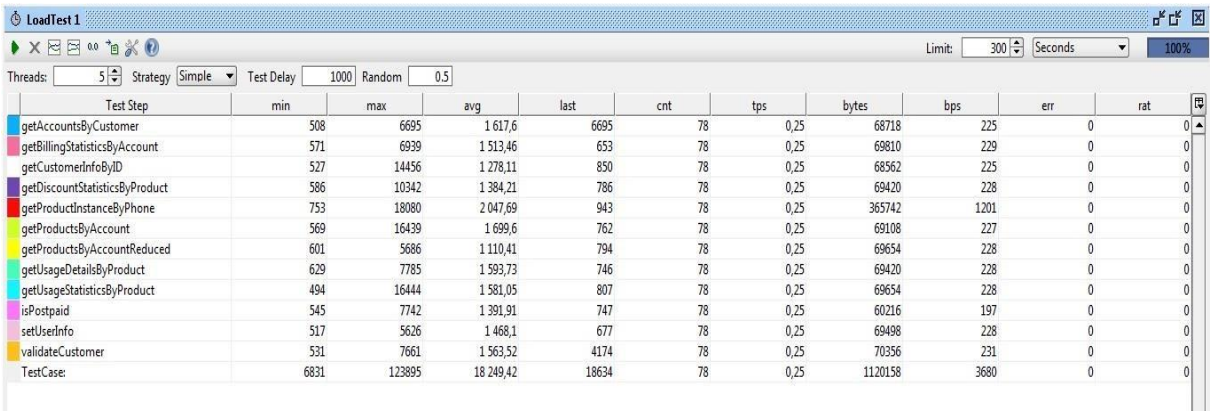
```
[citrus] 4547   INFO  port.LoggingReporter| TEST FINISHED: GreetingJmsTest
[citrus] 4547   INFO  port.LoggingReporter| ------------------------------------------
[citrus] 4563   INFO  port.LoggingReporter| FINISH TESTSUITE citrus-samples-greeting
[citrus] 4563   INFO  port.LoggingReporter| ------------------------------------------
[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; чи перевіряє відповідь xml відповідність даному визначенню схеми (xsd).

На рисунку 3.31 наведено результати тестування інтеграції за допомогою soapUI.

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 швидкість передачі даних між системами визначається в інтервалах: до 30 с — висока, 30 - 50 — середня, від 50 — низька.



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 тестування навантаження Wizard API

Таким чином, на даному прикладі була продемонстрована можливість приймального тестування показати, що всі модулі інтегровані між собою і мають гарну взаємодію, при роботі не виявлено дефекти, отже система є відмовостійкою. І це підтверджується досвідченим шляхом.

3.6 Аналітичні показники тестування безпеки

КС 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 КС 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 виконання аналізу захищеності КС 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 підсумкова вибірка низки російських і зарубіжних компаній із різних галузей економіки (з тих підприємств,

які дозволили використовувати знеособлені дані), у своїй більшу частину склали промислові, фінансові та транспортні підприємства (рис. 3.32).

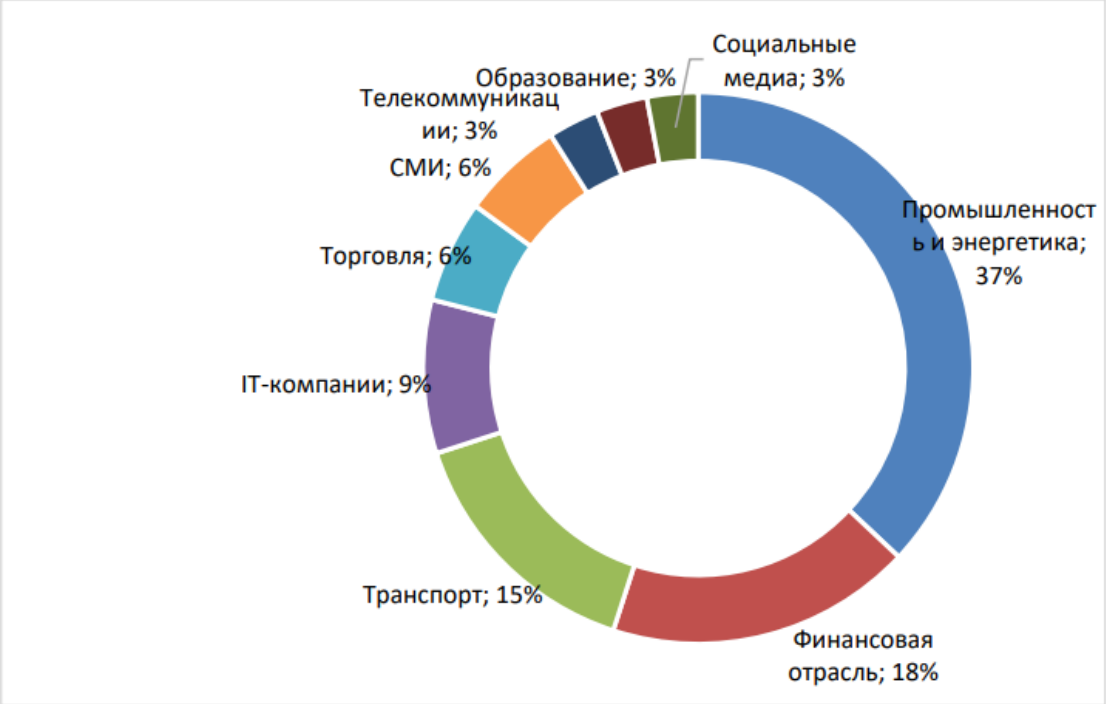


Рисунок 3.32 — Розподіл досліджених систем по галузях економіки

Аналіз захищеності КС 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.

Рисунок 3.33 — Види проведених робіт

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

— недоліки конфігурації;

— відсутність оновлень безпеки;

— вразливості у коді веб-додатків;

— недоліки парольної політики.

Кожній вразливості 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 (частка систем)

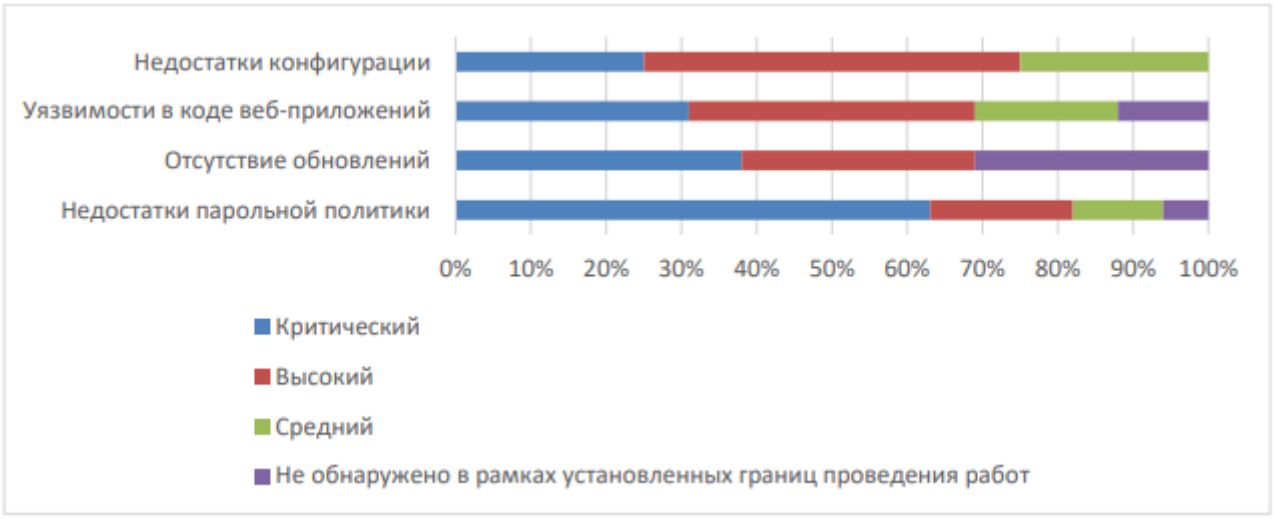


Рисунок 3.35 — Максимальний рівень небезпеки вразливостей (частка систем)

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. В середньому на одну систему припадає два вектори, а максимальна кількість векторів проникнення, виявлених в одній системі - п'ять. 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.

Рисунок 3.36 — Найкоротший шлях подолання мережевого периметра (частка систем)

Три чверті векторів пов'язані з недостатнім захистом веб-додатків: це основна проблема на мережному периметрі. При цьому якщо вектор складається з декількох кроків, на кожному кроці можуть експлуатуватись уразливості різного типу. Типовий сценарій атаки є підбір словникового облікового запису користувача КС і подальша експлуатація вразливості, що виникла через помилки в коді програми, наприклад можливості завантаження на сервер довільних файлів.

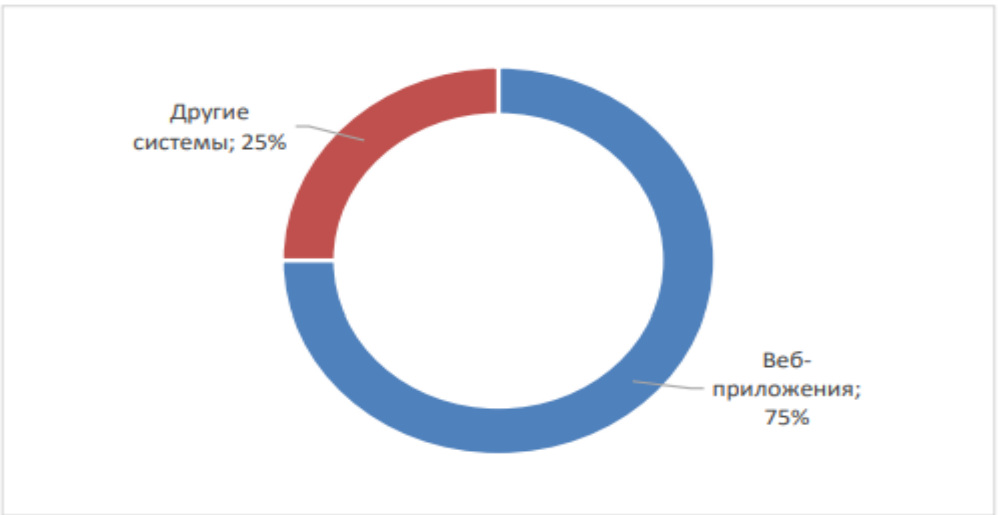


Рисунок 3.38 — Вектори проникнення у внутрішню мережу

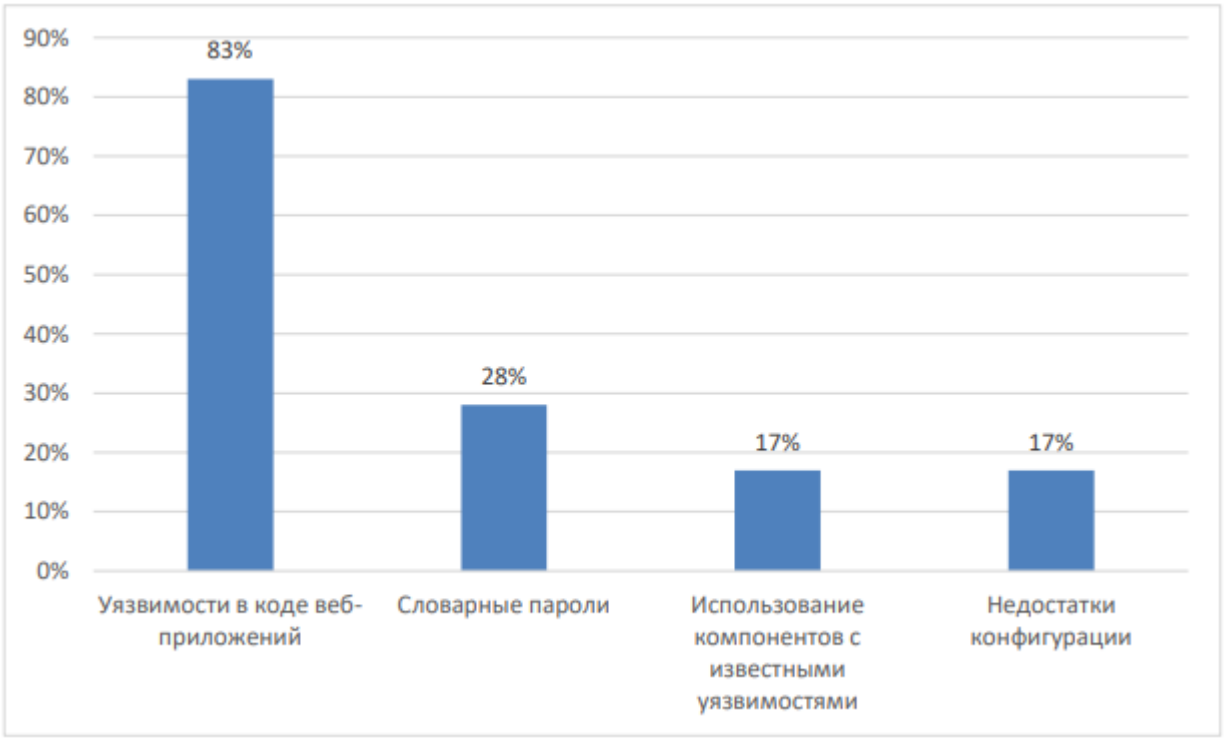


Рисунок 3.37 — Уразливості веб-додатків, що дозволили подолати мережевий периметр (частки векторів)

Рекомендується регулярно проводити аналіз захищеності веб-додатків. 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 рамках 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) та за протоколом RDP отримати доступ до комп'ютера співробітника компанії та внутрішньої мережі.

Для повного представлення тестування вразливості КС 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 найбільш поширених уразливостей 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. Вразливості можуть з'явитися будь-якої миті, оскільки конфігурація інфраструктури постійно змінюється, в ній з'являються нові вузли, нові системи та не виключені помилки адміністрування;

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

Рисунок 3.39 — Найбільш поширені вразливості на мережевому периметрі (частка систем)

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 уникнути виявлених загроз, 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 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 доборі 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 (ОС). 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, у 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). Хорошою практикою є використання двофакторної автентифікації.

Також як рекомендації пропонується відключити протоколи канального і 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.

Рисунок 3.40 — Найбільш поширені вразливості внутрішньої мережі (частка систем)

КС 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 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 найефективнішим методом перевірки є метод білого ящика, що передбачає аналіз вихідного коду. В якості превентивної міри рекомендується

використовувати міжмережевий екран рівня додатків (web applicationfirewall) для запобігання експлуатації вразливостей, які можуть з'являтися під час внесення змін до коду або додавання нових функцій. В рамках тестування на проникнення дії експертів рідко виявляються службою безпеки компаній, отже, і реальні зловмисники можуть довге 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 КС 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.

Важливо дотримуватися всіх рекомендацій у комплексі, оскільки навіть окремі прогалини у механізмах захисту можуть спричинити злом інфраструктуру та компрометацію критично важливих ресурсів. Рекомендується регулярно проводити тестування на проникнення, щоб виявляти вектори атак на КС та на практиці оцінювати ефективність вжитих заходів захисту.

4 АНАЛІЗ ЕФЕКТИВНОСТІ ЗАСТОСУВАННЯ ТЕСТУВАННЯ ВІДМОВОСТІЙКОСТІ КОМПЮТЕРНОЇ СИСТЕМИ

4.1 Оцінка обізнаності про рівень уразливості комп'ютерної системи

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 у фальшиву форму автентифікації. Помітна частка користувачів (14%) розкриває конфіденційну інформацію в розмові по телефону або листується з умовним зловмисником, повідомивши при цьому додаткові відомості про компанію: імена та посади співробітників, номери внутрішніх та мобільних телефонів (рис. 4.1).

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 є потенційним вектором проникнення у внутрішню інфраструктуру компанії. 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 з восьми протестованих систем бездротові мережі доступні за межами контрольованої зони, а отже, 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.

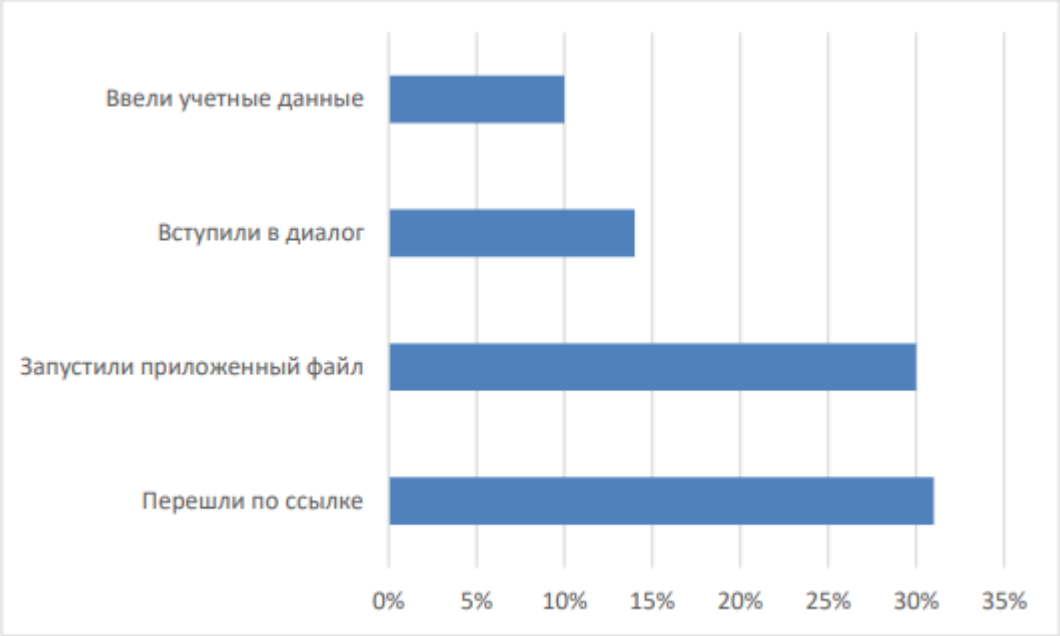


Рисунок 4.1 — Результати оцінки обізнаності співробітників

Залежно від використовуваного методу аутентифікації 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. Під час перевірки 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 4.2 — Методи аутентифікації у бездротових мережах

Результат атаки залежить від того, наскільки співробітники компанії обізнані з питань інформаційної безпеки.

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

4.2 Аналіз ефективності комплексного тестування комп'ютерної системи

Перед будь-яким інтеграційним процесом завжди стояло завдання покращення роботи комплексу КС.

Як критерії оцінки добре побудованого інтеграційного процесу можна виділити такі:

— швидкість передачі даних;

— ефективність обміну даними;

— оцінка користувачів інтеграційних систем

На рисунку 4.3 представлена гістограма, що відображає різницю швидкості роботи між інтеграційними операціями до і після апробації сценарію тестування.

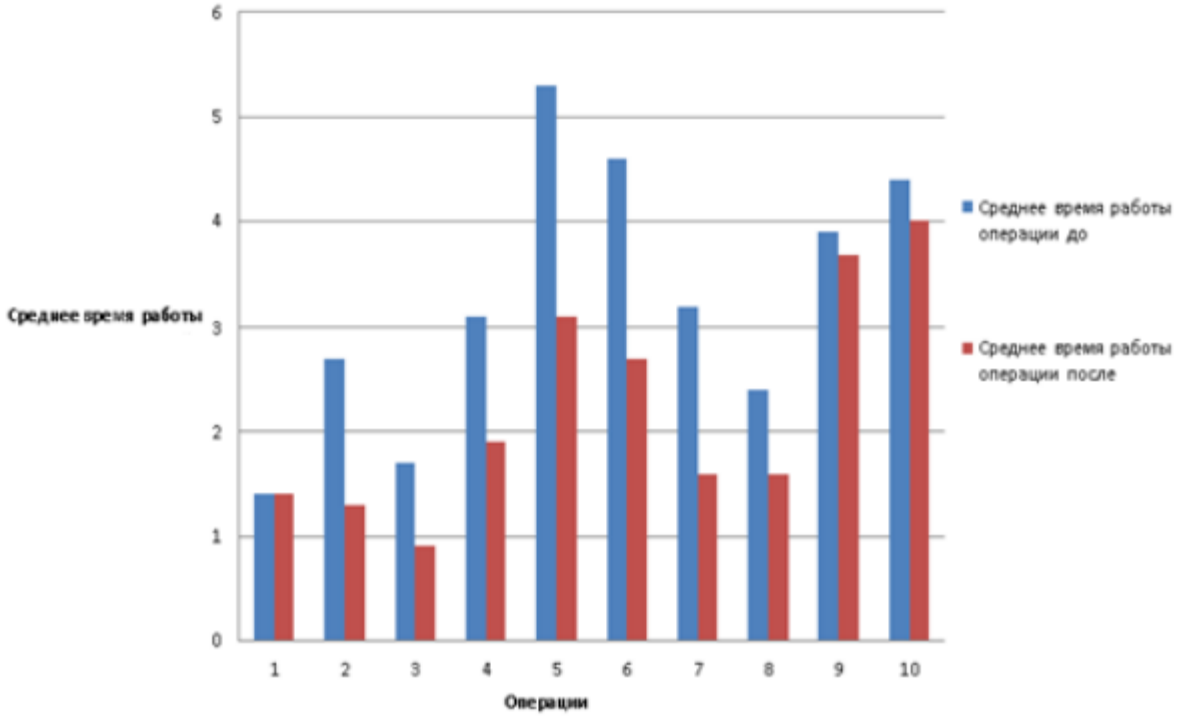


Рисунок 4.3 – Середній час роботи інтеграційних операцій

За даними гістограми можна зробити висновок, що запропонований сценарій тестування, що включає методи комплексного тестування, сприяє підвищенню якості роботи КС, але для чистоти експерименту, щоб оцінити достовірність результатів дослідження, необхідно скористатися критеріями ефективності.

Висунемо основну (нульову) гіпотезу H0 і перевіримо, чи не суперечить вона наявним емпіричним даним. Конкуруючою (альтернативною) гіпотезою призначимо H1, яка суперечить нульовій.

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 рівною кількістю значень у вибірках. Критерій t Стьюдента спрямований на оцінку відмінностей середніх величин x і y двох вибірок X та Y, які розподілені за нормальним законом. Однією з головних переваг критерію є широта його застосування.

Як правило, в дослідженнях найчастіше застосовують такі параметричні критерії як: t-критерій Стьюдента, що дозволяє оцінювати відмінності середніх у

двох вибірках і F — критерій Фішера, який оцінює відмінності між двома дисперсіями. Вони дозволяють отримати найбільш наочне уявлення аналізованих сукупностей. Обчислення значення $t_{емп}$ здійснюється за формулою:

$$t_{емп} = \frac{d}{Sd} \tag{4.1}$$

де

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

де

$di = xi - yi$ — 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 (рис. 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.

	Переменная 1	*Переменная 2*
Среднее	3,27	2,22
Дисперсия	1,626777778	1,166222222
Наблюдения	10	10
Корреляция Пирсона	0,817653538	
Гипотетическая разность средних	0	
df	9	
t-статистика	4,516158041	
P(T<=t) одностороннее	0,000727396	
t критическое одностороннее	1,833112933	
P(T<=t) двухстороннее	0,001454792	
t критическое двухстороннее	2,262157163	

Рисунок 4.4 — Результат розрахунку процедури «Парний двовибірковий t-тест для середніх»

За допомогою процедури «Парний двовибірковий t-тест для середніх» перевіримо на рівні значущості $\alpha = 0.05$ гіпотезу H1: середні часові інтервали процесів обміну даними між системами із застосуванням запропонованої методики комплексного тестування суттєво зменшилися. Основна гіпотеза H0: середній час процесів обміну даними між системами без застосування запропонованої методики та з її застосуванням суттєво не змінилося.

Аналіз результатів рішення показав, що розрахункове значення $t = 4,5$ статистики T перебуває у області $(2,26; +\infty)$. Це означає, що гіпотеза H0 про рівність часу роботи інтеграційних операцій без застосування запропонованої методики та з її застосуванням суперечить фактичним даним спостереження і, отже, її треба відхилити (на рівні значущості $\alpha = 0.05$) та прийняти альтернативну гіпотезу H1 що передбачає, що середній час процесів обміну даними між модулями КС без застосування запропонованої методики тестування більше часу роботи після застосування. Таким чином, початкове припущення підтвердилося, дійсно, середній час роботи інтеграційних операцій, після впровадження сценарію комплексного тестування КС дає кращі результати порівняно з попередніми.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Комерційний та технологічний аудит науково-технічної розробки

Метою даного розділу є проведення технологічного аудиту, в даному випадку комп'ютеризованих засобів тестування відмовостійкості елементів комп'ютерної системи. Особливістю розробки є розробка сценарію комплексного тестування з ціллю підвищення продуктивності та безпеки комп'ютерних систем в цілому. Наукова новизна полягає урозроблені засобів комплексного тестування комп'ютерних систем, що дозволяє на основі визначення типів помилок, що найчастіше зустрічаються, задавати якісні тести, що забезпечують мінімізацію всіляких втрат у бізнес-середовищі організації.

Аналогів розробки є декілька:

1. NeoLoad, платформа для тестування відмовостійкості, розроблена для DevOps, яка легко інтегрується у існуючий конвеєр безперервної доставки за ціною 10000 $/360000 грн.

2. LoadView Testing-платформа для тестування інфраструктури у будь-якому масштабі за ціною 14388 $/518000 грн.

3. HP LoadRunner – це найпопулярніший на сьогоднішній день інструмент для тестування відмовостійкості, який має віртуальний генератор користувачів, що імітує дії живих користувачів за ціною 10500 $/378000 грн.

4. Jmeter-один з провідних інструментів, що використовуються для тестування навантаження веб-серверів і серверів додатків за ціною 12000$ /432000 грн.

Для проведення комерційного та технологічного аудиту залучають не менше 3-х незалежних експертів. Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням п'ятибальної системи оцінювання за 12-ма критеріями, у відповідності із табл. 4.1.

Таблиця 5.1 — Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Бали (за 5-ти бальною шкалою)					
Крите-рій	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздат-ність продук-ту в реаль-них умовах
Ринкові переваги					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно до-рівнює цінам	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в ана-логів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Техніч ні та споживчі властивості продукту значно
5	Експлуата ційні витрати значно вищі, ніж в аналогів	Експлуата ційні витрати дещо вищі, ніж в аналогів	Експлуата ційні витрати на рівні експлуа-таційних витрат аналогів	Експлуата ційні витрати трохи нижчі, ніж в аналогів	Експлу атаційні витрати значно нижчі, ніж в
Ринкові перспективи					
6	Ринок малий і не має позити-вної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих ком-паній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкуренті в немає
Практик на здійсненність					
8	Відсутні фахівці як з технічної, так і з ко-мерційної реа-лізації ідеї	Необхідно на-ймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне не-значне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так із комерційної реалізації ідеї

Продовження табл. 5.1 — Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Усі дані по кожному параметру занесено в таблиці 5.2

Таблиця 5.2 — Результати оцінювання комерційного потенціалу розробки

Критерії оцінювання	ПІБ експертів		
	Експерт 1	Експерт 2	Експерт 3
	Бали		
Технічна здійсненність концепції	4	4	4
Наявність аналогів на ринку	3	4	4
Цінова політика	4	4	4

Технічні та споживчі властивості виробу	4	3	4
Експлуатаційні витрати	4	4	4
Ринок збуту	4	3	4
Конкурентоспроможність	3	4	3

Продовження табл. 5.2 — Результати оцінювання комерційного потенціалу розробки

Фахівці з технічної і комерційної реалізації	4	3	4
Фінансування	4	4	3
Матеріально-технічна база	3	3	3
Термін реалізації ідеї	4	4	4
Супровідна документація	4	4	4
Сума	45	44	45
Середньоарифметична сума балів	(45+44+45) / 3 = 44,67		

За даними таблиці 5.2 можна зробити висновок щодо рівня комерційного потенціалу даної розробки. Для цього доцільно скористатись рекомендаціями, наведеними в таблиці 5.3.

Таблиця 5.3 - Рівні комерційного потенціалу розробки

Середньоарифметична сума балів, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0-10	Низький
11-20	Нижче середнього
21-30	Середній
31-40	Вище середнього
41-48	Високий

Як видно з таблиці, рівень комерційного потенціалу розроблюваного нового програмного продукту є високим, що досягається за рахунок розробки сценарію комплексного тестування з ціллю підвищення продуктивності та безпеки комп'ютерних систем в цілому. Наукова новизна полягає у розроблені засобів комплексного тестування комп'ютерних систем, що дозволяє на основі визначення типів помилок, що найчастіше зустрічаються, задавати якісні тести, що забезпечують мінімізацію всіляких втрат у бізнес-середовищі організації.

5.2 Прогнозування витрат на виконання науково-дослідної (дослідно-конструкторської) роботи

5.2.1 Основна заробітна плата розробників, яка розраховується за формулою:

$$З_о = \frac{M}{T_р} \cdot t, \quad (5.1)$$

де М – місячний посадовий оклад конкретного розробника (дослідника), грн.;

$Т_р$ – число робочих днів за місяць, 22 днів;

t – число днів роботи розробника (дослідника).

Результати розрахунків зведемо до таблиці 5.4.

Таблиця 5.4 – Основна заробітна плата розробників

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
Керівник проекту	54000	2454,55	34	83454,545
Програміст	46800	2127,27	34	72327,273
Всього				155781,82

Так як в даному випадку розробляється програмний продукт, то розробник виступає одночасно і основним робітником, і тестувальником розроблюваного програмного продукту.

5.2.2 Додаткова заробітна плата розробників, які брати участь в розробці обладнання/програмного продукту.

Додаткову заробітну плату прийнято розраховувати як 13,8 % від основної заробітної плати розробників та робітників:

$$З_д = З_о \cdot 13{,}8\ \% / 100\ \% \quad (5.2)$$

$$З_д = (155781{,}82 \cdot 13{,}8\ \% / 100\ \%) = 21497{,}89 \text{ (грн.)}$$

5.2.3 Нарахування на заробітну плату розробників.

Згідно діючого законодавства нарахування на заробітну плату складають 22 % від суми основної та додаткової заробітної плати.

$$Н_з = (З_о + З_д) \cdot 22\ \% / 100\% \qquad (5.3)$$

$Н_з = (155781{,}82 + 21497{,}89) \cdot 22\ \% / 100\ \% = 39001{,}54$ (грн.)

5.2.4. Оскільки для розроблювального пристрою не потрібно витрачати матеріали та комплектуючи, то витрати на матеріали і комплектуючі дорівнюють нулю.

5.2.5 Амортизація обладнання, яке використовувалось для проведення розробки.

Амортизація обладнання, що використовувалось для розробки в спрощеному вигляді розраховується за формулою:

$$A = \frac{Ц}{Тв} \cdot \frac{t_{вик}}{12} \text{ [грн.].} \qquad (5.4)$$

де Ц – балансова вартість обладнання, грн.;

Т – термін корисного використання обладнання згідно податкового законодавства, років

$t_{вик}$ – термін використання під час розробки, місяців

Розрахуємо, для прикладу, амортизаційні витрати на комп’ютер балансова вартість якого становить 89938 грн., термін його корисного використання згідно податкового законодавства – 2 роки, а термін його використання — 1,55 міс.

$$A_{обл} = \frac{89938}{2} \times \frac{1{,}55}{12} = 5791{,}46\ грн.$$

Аналогічно визначаємо амортизаційні витрати на інше обладнання та

приміщення. Розрахунки заносимо до таблиці 4.5. Так як вартість ліцензійної ОС та спеціалізованих ліцензійних нематеріальних ресурсів менше 20000 грн. (Microsoft Windows Server 2008 R2 x64, Ubuntu Server 12.04 64-бітна), то даний нематеріальний актив не амортизується, а його вартість включається у вартість розробки повністю, $В_{нем.ак.}$ = 18900 грн.

Таблиця 5.5 — Амортизаційні відрахування на матеріальні та нематеріальні ресурси для розробників

Найменування обладнання	Балансова вартість, грн.	Строк корисного використання, років	Термін використання обладнання, місяців	Амортиза-ційні відрахуван-ня, грн.
Комп'ютер та комп'ютерна периферія (Asus RT-AC68U, Xeon E5-2630, BLOB у SharePoint Foundation, Smart-UPS RT 3000VA, Dell Precision 7510 (4K IGZO TP-LINK TL-WA901ND, Canon i-SENSYS MF3010)	89938	2	1,55	5791,462
Офісне обладнання (меблі)	30000	4	1,55	965,909
Приміщення	1300000	20	1,55	8371,212
Всього				15128,58

5.2.6 Тарифи на електроенергію для непобутових споживачів (промислових підприємств) відрізняються від тарифів на електроенергію для населення. При цьому тарифи на розподіл електроенергії у різних постачальників (енергорозподільних компаній), будуть різними. Крім того, розмір тарифу залежить від класу напруги (1-й або 2-й клас). Тарифи на розподіл електроенергії для всіх енергорозподільних компаній встановлює Національна комісія з регулювання енергетики і комунальних послуг (НКРЕКП). Витрати на силову електроенергію розраховуються за формулою:

$$В_е = В \cdot П \cdot Ф \cdot К_п, \qquad (5.5)$$

де В – вартість 1 кВт-години електроенергії для 1 класу підприємства, В = 6,2 грн./кВт;

П – встановлена потужність обладнання, кВт. П = 0,5 кВт;

Ф – фактична кількість годин роботи обладнання, годин.

$К_п$ – коефіцієнт використання потужності, $К_п$ = 0,9.

$$В_е = 0,9 \cdot 0,5 \cdot 8 \cdot 34 \cdot 6,2 = 758,88 \text{ (грн.)}$$

5.2.7 Інші витрати та загальновиробничі витрати.

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками. Витрати за статтею «Інші витрати» розраховуються як 50…100% від суми основної заробітної плати дослідників:

$$I_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (5.6)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати».

$$І_в = 155781,82 * 85\% / 100\% = 132414,5 \text{ (грн.)}$$

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов’язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов’язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов’язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін. Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуються як 100…150% від суми основної заробітної плати дослідників:

$$H_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (5.7)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати».

$$Н_{нзв} = 155781,82 * 115\ \% / 100\ \% = 179149 \text{ (грн.)}$$

5.2.9 Витрати на проведення науково-дослідної роботи.

Сума всіх попередніх статей витрат дає загальні витрати на проведення науково-дослідної роботи:

$$В_{заг} = 155781,82+21497,89+39001,54+15128,58+18900+758,88+132414,5+ +179149 = 562632,34 \text{ грн.}$$

5.2.11 Розрахунок загальних витрат на науково-дослідну (науково-технічну) роботу та оформлення її результатів.

Загальні витрати на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховуються за формулою:

$$ЗВ = \frac{В_{заг}}{\eta} \text{ (грн)}, \quad (5.8)$$

де η – коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи.

Так, якщо науково-технічна розробка знаходиться на стадії: науково-дослідних робіт, то η=0,1; технічного проектування, то η=0,2; розробки конструкторської документації, то η=0,3; розробки технологій, то η=0,4; розробки дослідного зразка, то η=0,5; розробки промислового зразка, то η=0,7;

впровадження, то η=0,9. Оберемо η = 0,5, так як розробка, на даний момент, знаходиться на стадії дослідного зразка:

$$ЗВ = 562632,34 / 0,5 = 1125265 \text{ грн.}$$

5.3 Розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором

В ринкових умовах узагальнювальним позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку. Саме зростання чистого прибутку забезпечить потенційному інвестору надходження додаткових коштів, дозволить покращити фінансові результати його діяльності, підвищить конкурентоспроможність та може позитивно вплинути на ухвалення рішення щодо комерціалізації цієї розробки.

Для того, щоб розрахувати можливе зростання чистого прибутку у потенційного інвестора від можливого впровадження науково-технічної розробки необхідно:

а) вказати, з якого часу можуть бути впроваджені результати науково-технічної розробки;

б) зазначити, протягом скількох років після впровадження цієї науково-технічної розробки очікуються основні позитивні результати для потенційного інвестора (наприклад, протягом 3-х років після її впровадження);

в) кількісно оцінити величину існуючого та майбутнього попиту на цю або аналогічні чи подібні науково-технічні розробки та назвати основних суб’єктів (зацікавлених осіб) цього попиту;

г) визначити ціну реалізації на ринку науково-технічних розробок з аналогічними чи подібними функціями.

При розрахунку економічної ефективності потрібно обов'язково враховувати зміну вартості грошей у часі, оскільки від вкладення інвестицій до отримання прибутку минає чимало часу. При оцінюванні ефективності інноваційних проектів передбачається розрахунок таких важливих показників:

— абсолютного економічного ефекту (чистого дисконтованого доходу);

— внутрішньої економічної дохідності (внутрішньої норми дохідності);

— терміну окупності (дисконтованого терміну окупності).

Аналізуючи напрямки проведення науково-технічних розробок, розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором можна об'єднати, враховуючи визначені ситуації з відповідними умовами.

5.3.1 Розробка чи суттєве вдосконалення програмного засобу (програмного забезпечення, програмного продукту) для використання масовим споживачем

В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

$$\Delta П_i = (\pm \Delta Ц_0 \cdot N + Ц_0 \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \qquad (5.9)$$

де $\pm\Delta Ц_о$ – зміна вартості програмного продукту (зростання чи зниження) від впровадження результатів науково-технічної розробки в аналізовані періоди часу;

N – кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки;

$Ц_о$ – основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки, $Ц_о = Ц_б \pm \Delta Ц_о$;

Цб – вартість програмного продукту у році до впровадження результатів розробки;

ΔN – збільшення кількості споживачів продукту, в аналізовані періоди часу, від покращення його певних характеристик;

λ – коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт λ = 0,8333.

р – коефіцієнт, який враховує рентабельність продукту;

ϑ – ставка податку на прибуток, у 2022 році ϑ =18%.

Припустимо, що при прогнозованій ціні 75000 грн. за одиницю виробу, термін збільшення прибутку складе 3 роки. Після завершення розробки і її вдосконалення, можна буде підняти її ціну на 5000 грн. Кількість одиниць реалізованої продукції також збільшиться: протягом першого року – на 1800 шт., протягом другого року – на 1200 шт., протягом третього року на 1000 шт. До моменту впровадження результатів наукової розробки реалізації продукту не було:

$\Delta П_1$ = (0*5000 + (75000 + 5000)*1800)* 0,8333* 0,15) * (1 - 0,18) = 13837499,447 грн.

$\Delta П_2$ = (0*5000 + (75000 + 5000)*(1800+1200)* 0,833* 0,15) * (1 - 0,18) = 24599999,01 грн.

$\Delta П_3$ =(0*5000+(75000+5000)*(1800+1200+1000)* 0,833*0,15) *(1 - 0,18)= 32799998,68 грн.

Отже, комерційний ефект від реалізації результатів розробки за три роки складе 71237497,15 грн.

5.3.2 Розрахунок ефективності вкладених інвестицій та періоду їх окупності

Розраховуємо приведену вартість збільшення всіх чистих прибутків ПП, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \qquad (5.10)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої науково-дослідної (науково-технічної) роботи, грн;

Т – період часу, протягом якою виявляються результати впровадженої науково-дослідної (науково-технічної) роботи, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ = 0,05…0,15;

t – період часу (в роках).

Збільшення прибутку ми отримаємо, починаючи з першого року:

$$ПП = (13837499,447/(1+0,1)^1)+(24599999,016/(1+0,1)^2)+(32799998,688/$$
$$/(1+0,1)^3) = 12579544,95 + 20330577,7 + 24643124,48 = 57553247,13 \text{ грн.}$$

Далі розраховують величину початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки. Для цього можна використати формулу:

$$PV = k_{інв} * ЗВ, \quad (5.11)$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо; зазвичай $k_{інв}$=2...5, але може бути і більшим;

ЗВ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, грн.

$$PV = 2 * 1125265 = 2250529,38 \text{ грн.}$$

Тоді абсолютний економічний ефект $Е_{абс}$ або чистий приведений дохід (NPV, Net Present Value) для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$Е_{абс} = ПП – PV, \quad (5.12)$$

$$Е_{абс} = 57553247,13 – 2250529,38 = 55302717,76 \text{ грн.}$$

Оскільки $E_{абс} > 0$ то вкладання коштів на виконання та впровадження результатів даної науково-дослідної (науково-технічної) роботи може бути доцільним.

Для остаточного прийняття рішення з цього питання необхідно розрахувати внутрішню економічну дохідність або показник внутрішньої но-рми дохідності (IRR, Internal Rate of Return) вкладених інвестицій та порівняти її з так званою бар'єрною ставкою дисконтування, яка визначає ту мінімальну внутрішню економічну дохідність, нижче якої інвестиції в будь-яку науково-технічну розробку вкладати буде економічно недоцільно.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $E_в$. Для цього використаємо формулу:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1, \tag{5.13}$$

$T_ж$ – життєвий цикл наукової розробки, роки.

$$Е_в = \sqrt[3]{(1 + 55302717,76/2250529,38} - 1 = 1,946$$

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою:

$$\tau = d + f, \tag{5.14}$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2022 році в Україні d = (0,09…0,14);

f – показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,5).

$$\tau_{\min} = 0{,}14 + 0{,}05 = 0{,}19 .$$

Так як $E_в > \tau_{min}$, то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій за формулою:

$$T_{ок} = \frac{1}{E_в}, \qquad (5.15)$$

$$T_{ок} = 1 / 1{,}946 = 0{,}51 \text{ р.}$$

Оскільки $T_{ок}$ < 3-х років, а саме термін окупності рівний 0,51 роки, то фінансування даної наукової розробки є доцільним.

Висновки до розділу: економічна частина даної роботи містить розрахунок витрат на розробку нового програмного продукту, сума яких складає 1125265 гривень. Було спрогнозовано орієнтовану величину витрат по кожній з статей витрат. Також розраховано чистий прибуток, який може отримати виробник від реалізації нового технічного рішення, розраховано період окупності витрат для інвестора та економічний ефект при використанні даної розробки. В результаті аналізу розрахунків можна зробити висновок, що розроблений програмний продукт за ціною дешевший за аналог і є висококонкурентоспроможним. Період окупності складе близько 0,51 роки.

ВИСНОВКИ

У ході проведеного дослідження було розглянуто найбільш популярні методи тестування, які застосовуються для проведення комплексного тестування відмовостійкості елементів комп'ютерних систем. Показано, що комплексне тестування КС відіграє важливу роль у процесі її життєвого циклу для визначення її працездатності, надійності та відмовостійкості.

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

Розглянуто теоретичні засади тестування комп'ютерних систем, що дозволяють здійснювати аналіз наявних методів тестування відмовостійкості елементів комп'ютерних систем.

Проаналізовано з урахуванням розробленої теорії практичні методи тестування відмовостійкості програмного забезпечення комп'ютерних систем.

Розроблено сценарій комплексного тестування відмовостійкості, що дозволяє з одного боку оцінити продуктивність, з другого — її інформаційну безпеку.

Підтверджено шляхом експерименту ефективність реалізованого сценарію та його результати.

Враховуючи запропоновану методику (сценарій) комплексного тестування відмовостійкості КС можна використовувати для оцінки її надійності та ефективності в будь-який момент її функціонування.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. ГОСТ 19.701-90. Схеми алгоритмів, програм, даних та систем. Умовні позначення та правила виконання (ІСО 5807-85). Введ. 1992-01-01.
– М.: Видавництво стандартів, 1992 - 14 с. – (Єдина система програмної документації).

2. ГОСТ Р ІСО/МЕК 12207-99. Інформаційна технологія. Процеси життєвого циклу програмних засобів. Введ. 2000-07-01. - М.: Видавництво стандартів, 2000. - 30 с.

3. ГОСТ 7.82-2001. Бібліографічний запис. Бібліографічне опис електронних ресурсів. Загальні вимоги та правила складання. - Введ. 2002-07-01. - Мінськ: Видавництво стандартів, 2001. - 35 с. – (Система стандартів з інформації, бібліотечної та видавничої справи).

4. ГОСТ Р ИСО/МЭК 9126-93 “Інформаційна технологія. Оцінка програмної продукції. Характеристики якості та посібника щодо їх застосування”

5. Дастін Е. Тестування програмного забезпечення. Впровадження, управління та автоматизація / Е. Дастін, Д. Решка, Д. Пол; Пров. з англ. М. Павлов. – К.: Лорі, 2013. – 567 с.

6. Котляров В.П. Основи сучасного тестування програмного забезпечення: навчальний посібник / В.П.Котляров, Т.В.Колікова - СПб.: Пітер, 2004.- 170 с.

7. Куликов С. С. Тестування програмного забезпечення. Базовий курс: практ. допомога. / С. С. Куликов. - Мінськ: Чотири чверті, 2015. - 294 с.

8. Липаєв В.В. Тестування великих комплексів програм відповідність вимогам: підручник/В.В.Липаев – М.: ІСЦ «Глобус», 2008. – 316 з.

9. Ліпаєв В.В., Позін Б.А., Блау С.А., Аналіз стратегій тестування логіки програм, Кібернетика, 1982, "2, з 45-50.

10. Методи аналізу даних // Молодий учений. - 2015.- №13. - С. 167-169.

11. Савін Р. Тестування Дот Ком, або посібник з жорстокого поводження з багами в інтернет-стартапах / Р. Савін - М.: Справа, 2007. - 312 с.

12. Савін Р. Тестування Дот Ком, або Посібник із жорстокого поводження з багами в інтернет-стартапах. - М.: Справа, 2007. - 312 с

13. Сергєєва Н.А. Стратегія тестування інформаційних систем управління вузом на основі документів із теговою розміткою // Вісник РУДН, серія Інформатизація освіти. – 2012. – № 3. – С. 31-37.

14. Сем Канер. Testing Computer Software/Сем Канер, Джек Фолк. - М.: ДіаСофт, 2001. - 544 с.

15. Філіппов В. А., Хатько Є. Є. Генерація тестових сценаріїв для мобільних додатків. Інформаційні, мережеві та телекомунікаційні технології. 2012.Т. 4.

16. Філіппов В. А., Хатько Є. Є. Проблемні питання автоматизації тестування для мультизадачних користувальницьких комплексів. Інформаційні, мережеві та телекомунікаційні технології. 2012 . Т. 4.

17. Хатько Є. Є. Сучасні проблеми фундаментальних та прикладних наук. Один спосіб реалізації алгоритму генерації тестів у тестуванні на основі моделей. Т. 1, с. 92-95. 52. М. 2010.

18. Хатько Є. Є., Філіппов, В. А. Проблеми якості тестування програмного забезпечення для мультизадачних користувальницьких комплексів. Якість. Інновації. Освіта. 3 2011.Т. 3, с. 32-35.

19. Шмейлін Б. 3. Сучасні технології тестування WEB додатків. Системи та засоби інформатики. 2009 р., с. 138-147.

20. Бородін А.М., Мирвода С.Г., Поршнєв С.В. Особливості тестування стійкості до збоїв корпоративних інформаційних систем методом генерування відмов // Сучасні проблеми науки та освіти. 2014. - № 5. URL:http://www.science-education.ru/ru/article/view?id=14997(Дата звернення: 07.03.2019).

21. Волков В.Г., Автоматизована система контролю та забезпеченнянадійності програмних засобів // http://www.unn.ru/pages/issues/vestnik/99999999_West_2009_5/27.pdf (дата звернення 25.02.2019)

22. Ключов А.О., Маковецька Н.А., Проблеми тестування системного інформаційного забезпечення розподілених інформаційно-керуючих систем //http://www.ict.edu.ru/ft/001795/vestnik10_7.pdf

23. Котов С.Л., Навантажувальне тестування як елемент формування безпечних систем// http://www.ooogic.ru/downloads/loading%20test%20as%20an%20element.pdf

24. Кулямін В. В. Тестування на основі моделей, URL:http://panda.ispras.ru/~kuliamin/lectures-mbt/Lecture04.pdf.(Дата звернення 05.02. 2019).

25. Петренко О. Тестування на основі моделей. Інформаційні системи. URL:http://www.info-system.ru/testing/testtestingbasismodels.html.(дата звернення 21.02. 2019)

26. Старолетів С.М., Крючкова О.М. Тестування розподілених додатків на основі побудови моделей [Електронний ресурс]: навчальнийпосібник/С.М.Старолетов,Е.Н.Крючкова.–Режим доступу: http://cyberleninka.ru/article/n/testirovanie-raspredelennyhprilozheniy-na-osnove-pobudovi-моделі

27. Технологія каскадного випробування програмного забезпечення. - Режим доступу:http://software-testing.ru/about/trainers/94-rukol

28. рівні тестування програмного забезпечення. - Режим доступу:http://www.protesting.ru/testing/testlevels.html

Література іноземною мовою

29. Andrews A., Offut J., Alexander R. Testing Web Applications by Modeling with FSMs. б.м. : National Science Foundation, 2005.

30. Changyou Xing,GuominZhang, Ming Chen. Research on universal network performance testing model/ International Symposium on Communications and Information Technologies, 2007. P. 780-784.

31. Heiskanen H., Maunumaa M., Katara, M. Test Process Improvement for Automated Test Generation. Tampere: Tampere University of Technology, Department of Software Systems, 2010.

32. Jie M., Honlin Zh., Wenbo X., Jin L., Reliability Testing Methods for Critical Information System заснований на State Random //http://www.ipcsit.com/vol16/6-ICICM2011M009.pdf

33. Khatko E, Fillipov V. Mobile applications testing processes metrics and optimization criteria. Software Engineering. 5, 2012, T. 2.

34. Kim G.-B. F метод генерування масивних віртуальних клієнтів і моделювання базових результатів тестування / Fifth International Conference on Quality Software, 2005. P. 250-254.

35. Костогризов А., В.Панов, Б.Позин, В.Саблін. Mathematical modeling processes в системах життя циклів в compliance with standards requirements ISO/IEC 15288 and ISO/IEC 12207, Spincose, Montreal, Canada, 2003.

36. Makinen M. Model Based Approach to Software. Helsinki: Helsinki University of Technology, 2007.

37. MSDN Magazine, microsoft.com. URL:http://msdn.microsoft.com/ra-ru/magazine/dd419663 .aspx. (Retrieved. 09.11.2018)

38. Robinson H. Graph Theory Techniques в Model-Based Testing. 1999.

39. The Next Generation 1996 Lexicon A to Z".Next Generation. No. 15.Imagine Media. Alpha software, як правило, barely runs і є missing major features як gameplay and complete levels. March 1996. p. 29.

40. Xijiang L., Pomeranz I., Sudhakar M. Techniques for Improving Efficiency of Sequential Circuit Test Generation. Iowa: University of Iowa. 2015 року.

ДОДАТОК А

Міністерство освіти та науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп’ютерної інженерії

Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ

Завідувач кафедри ОТ

_____проф., д.т.н. О. Д. Азаров

«___» _____ 2022 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи

«Засоби тестування відмовостійкості елементів комп’ютерної системи»
08-23.МКР.026.00.000 ПЗ

Науковий керівник: к. т. н., доц. каф ОТ

_____ Колесник І.С.

Магістрант групи 2КІ-21м

_____ Луценко Ю.В.

Вінниця 2022

1. Підстава виконання магістерської кваліфікаційної роботи

1.1 Дуже часто розробникам, менеджерам проектів, керівнику фірми необхідно вирішувати питання, пов'язані із скороченням витрат на виробництво ПП та підвищенням якості програмного забезпечення. Основним способом вирішення цих проблем є тестування відмово стійкості програмного забезпечення КС, так і комп’ютерної системи в цілому. Процес тестування включає вирішення питань не тільки технічного характеру (організація ефективного процесу тестування, визначення часу тестування, використання або невикористання інструментальних засобів і т.д.), але і питань економічного і психологічного характеру. Безумовно, виходячи з вищевикладеного, питання, які розглядаються у роботі, на сьогоднішній день є актуальними.

1.2 Наказ про затвердження теми магістерської кваліфікаційної роботи 205-А від 15.09.2022 р.

2 Мета і призначенням МКР

2.1 Метою роботи є є аналіз засобів тестування відмовостійкості та розробка сценарію комплексного.

2.2 Призначення розробки — виконання магістерської кваліфікаційної роботи.

3. Вихідні дані для виконання МКР

Вихідні дані для виконання МКР: середовища розробки: Microsoft Windows Server 2008 R2 x64, маршрутизатор Asus RT-AC68U; Сервер Xeon E5-2630, 12 x 2.60 ГГц, процесорів 2 оперативна пам'ять – 8 ГБ, пам'ять – 2 x 500 ГБ, зовнішнє сховище даних BLOB у SharePoint Foundation.

4. Вимоги до виконання МКР

МКР повинна задовольняти такі вимоги:

— дозволити дослідити теоретичні засади тестування комп’ютерних систем,;

— дозволити проаналізовати з урахуванням розробленої теорії практичні методи тестування відмовостійкості програмного забезпечення комп’ютерних систем;

— розробити сценарій комплексного тестування відмовостійкості;

— підтвердити шляхом експерименту ефективність реалізованого сценарію та його результати

5. Етапи МКР та очікувані результати

Етапи роботи та очікувані результати приведено в Таблиці А.1.

Таблиця А.1 — Етапи МКР

№ з/п	Назва етапів виконання магістерської роботи	Строк виконання етапів роботи	Примітка
1	Постановка задач роботи	15.09.22	
2	Методологія тестування комп’ютерних систем	15.09-16.09.22	
3	Засоби комплексного тестування комп’ютерної системи	16.09-24.09.22	
4	Тестування комп’ютерної системи та її продуктивності	24.09-04.10.22	
5	Тестування навантаження з використанням різних інструментальних засобів	04.10-14.10.22	
6	Аналітичні показники тестування безпеки	14.10-23.10.22	
7	Оцінка обізнаності про рівень уразливості комп’ютерної системи	24.10-31.10.22	
8	Аналіз ефективності тестування комп’ютерної системи	01.11-16.11.22	
9	Розрахунок економічної частини роботи	17.11-30.11.22	
10	Оформлення пояснювальної записки та ілюстративного матеріалу	01.12-06.12.22	
11	Аналіз виконання роботи, висновки, додатки	07.12-06.12.22	
12	Перевірка якості виконання магістерської роботи та усунення недоліків	15.12.21	

6 Матеріали, що подаються до захисту МКР

До захисту МКР подаються: пояснювальна записка МКР, ілюстративні та графічні матеріали, протокол попереднього захисту МКР на кафедрі, відзив наукового керівника, відзив опонента, протоколи складання державних екзаменів,

анотації до МКР українською та іноземною мовами, довідка про відповідність оформлення МКР діючим вимогам.

7 Порядок контролю виконання та захисту МКР

Виконання етапів розрахункової та графічної документації МКР контролюється науковим керівником згідно зі встановленими термінами. Захист МКР відбувається на засіданні Державної екзаменаційної комісії, затвердженою наказом ректора.

8 Вимоги до оформлення МКР

При оформлювання МКР використовуються:

— ДСТУ 3008: 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302: 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— міждержавний ГОСТ 2.104-2006 «Едина система конструкторської документації. Основні написи»;

— Методичні вказівки до виконання магістерських кваліфікаційних робіт зі спеціальності 123 — «Комп’ютерна інженерія». Кафедра обчислювальної техніки ВНТУ 2022.

ДОДАТОК Б

Життєвий цикл тестування

Рисунок Б1 — Життєвий цикл тестування

ДОДАТОК В

Загальна схема основних видів тестування

Рисунок В1 — Загальна схема основних видів тестування

ДОДАТОК Д

Алгоритм процедури тестування за сценарієм 1

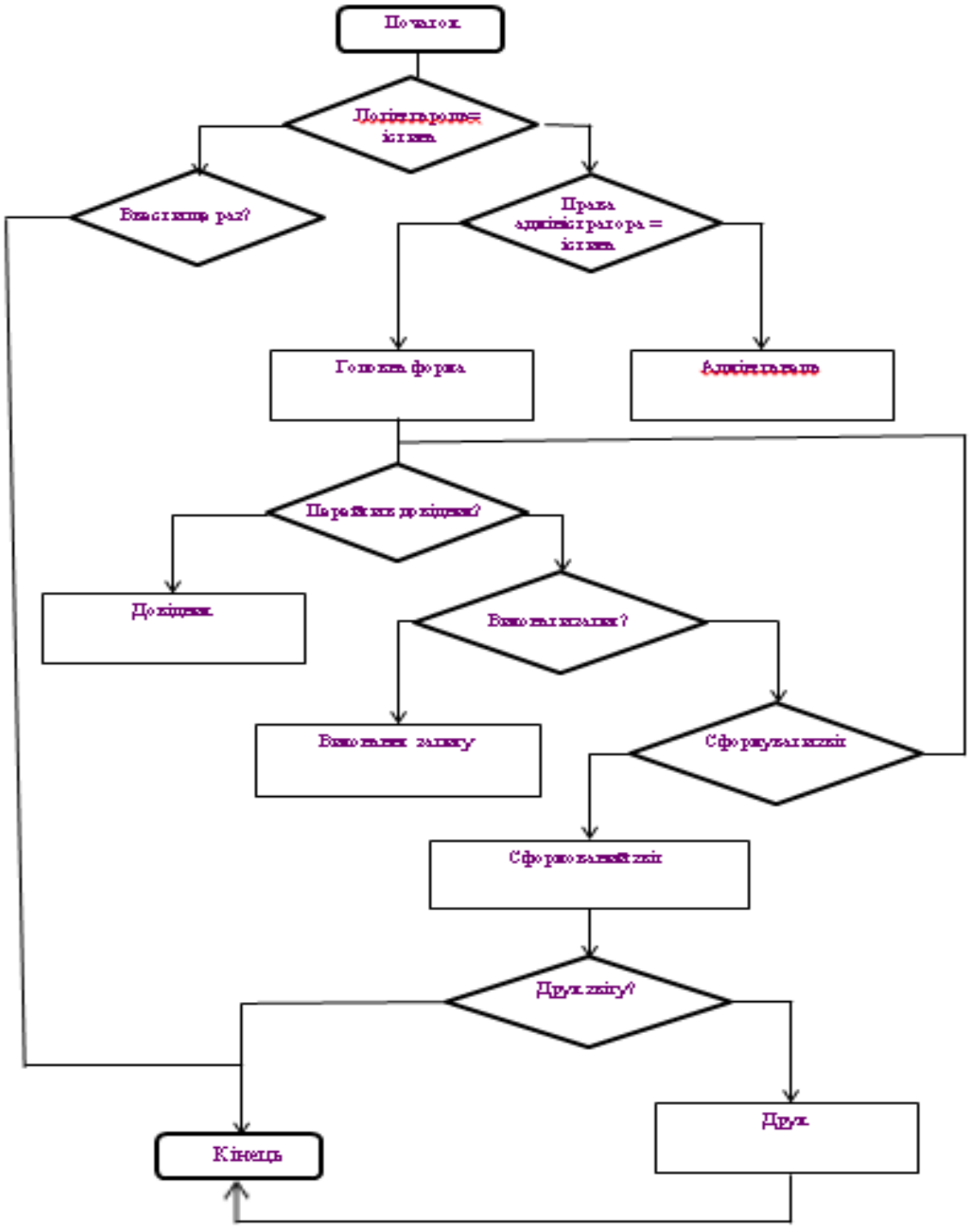


Рисунок Е1 — Алгоритм процедури тестування за сценарієм 1

ДОДАТОК Е
ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА
НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Засоби тестування відмовостійкості елементів комп'ю-терної системи

Тип роботи: магістерська кваліфікаційна робота
(БДР, МКР)

Підрозділ кафедра обчислювальної техніки
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 97.6% Схожість 2.4%

Аналіз звіту подібності (відмітити потрібне):

- ✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.
- ☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.
- ☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Захарченко С.М.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ _____
(підпис) (прізвище, ініціали)

Керівник роботи _____ _____
(підпис) (прізвище, ініціали)