Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА

Галузь знань 12 – Інформаційні технології

Спеціальність 126 – Інформаційні системи та технології

на тему: «Метод та засоби інформаційної технології ідентифікації залишкових дефектів у програмному забезпеченні»

КвРІСТ. 220173.22.01.01 ПЗ

Виконав: студент 2 курсу, група ІСТм-22-1 Засорнова І.О.
Підпис Ініціали, прізвище

Керівник: доктор техн. наук, професор Говорущенко Т.О.
Науковий ступінь, вчене звання Підпис Ініціали, прізвище

До захисту допускаю:
Зав. кафедри КІС, д.т.н., проф.
Т.О. Говорущенко
01 12 2023 р.

Хмельницький, 2023

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Комп'ютерної інженерії та інформаційних систем

Освітній рівень магістр

Галузь знань 12 Інформаційні технології

Спеціальність 126 Інформаційні системи та технології

Освітня програма освітньо-професійна програма «Інформаційні системи та технології»

ЗАТВЕРДЖУЮ

Зав. кафедри Т.О. Говорущенко

" 01 " 04 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ МАГІСТРА

Засорновій Ірині Олександрівні

Прізвище, ім'я, по батькові студента

1. Тема проєкту (роботи) Метод та засоби інформаційної технології ідентифікації залишкових дефектів у програмному забезпеченні

Керівник проєкту (роботи) Говорущенко Т.О., д.т.н., професор

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 15.08.2023 р. №30

2. Строк подання студентом проєкту (роботи) на кафедру 10.12.2023 р.

3. Вихідні дані до проекту (роботи) Завдання на дипломне проектування

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Аналіз відомих моделей, методів та засобів інформаційної т ехнології

Інформаційні потоки інформаційної технології ідентифікації залишкових дефектів у програмному забезпеченні

Метод ідентифікації залишкових дефектів у програмному забезпеченні

Інформаційна технологія ідентифікації залишкових дефектів у програмному забезпеченні

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

6. Консультанти розділів кваліфікаційної роботи магістра

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Лисенко С.М., професор кафедри КІС		
Антиплагіат	Нічепорук А.О., доцент кафедри КІС		

7. Дата видачі завдання «_03_»_____04_____2023р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) кваліфікаційної роботи магістра	Термін виконання етапів проекту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики КвРМ з керівником	03.04.2023	виконано
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	03.05.2023	виконано
3	Робота над розділом 1 – аналіз відомих моделей, методів за темою; постановка задачі	03.06.2023	виконано
4	Робота над розділом 2 – розробка моделей для вирішення поставленої задачі	03.07.2023	виконано
5	Робота над науковою статтею	01.10.2023	виконано
6	Робота над розділом 3 – розробка методів для вирішення поставленої задачі	01.10.2023	виконано
7	Робота над розділом 4 – проектування та розробка ПЗ для вирішення поставленої задачі, експериментальна частина	01.11.2023	виконано
8	Оформлення пояснювальної записки згідно вимог	15.11.2023	виконано
9	Попередній захист ДРМ	16.11.2023	виконано
10	Захист ДРМ на засіданні ЕК	До 20.12.2023	

Студент _____ Підпис І.О. Засорнова Ініціали, прізвище

Керівник роботи _____ Підпис Т.О. Говорущенко Ініціали, прізвище

РЕФЕРАТ

Тема кваліфікаційної роботи магістра: Метод та засоби інформаційної технології ідентифікації залишкових дефектів у програмному забезпеченні.

Автор роботи: Засорнова Ірина Олександрівна, студентка групи ІСТм-22-1.

Керівник роботи: Говорущенко Т.О., доктор технічних наук, професор, завідувач кафедри комп'ютерної інженерії та інформаційних систем.

Пояснювальна записка: 86 с., 30 рис., 6 табл., 5 дод., 74 джерела.

ПЕРЕЛІК КЛЮЧОВИХ СЛІВ: тестування програмного забезпечення, залишкові дефекти програмного забезпечення, інтелектуальна інформаційна технологія, ступінь критичності наслідків залишкових дефектів.

Об'єктом дослідження є процес виявлення залишкових дефектів у ПЗ.

Предметом дослідження є методи та засоби інформаційної технології виявлення залишкових дефектів у ПЗ.

Метою кваліфікаційної роботи є ідентифікація залишкових дефектів у ПЗ шляхом розробки методу та засобів інформаційної технології. Вивчаючи ці методи та засоби, можливо отримати уявлення про їх переваги та обмеження, а також зрозуміти їхню придатність для задач виявлення дефектів. Ці дослідження сприятимуть покращенню загального процесу розробки ПЗ.

Для розв'язання поставлених задач використовуються основні положення загальної теорії систем, системного аналізу (ієрархічності, декомпозиції та інші), теорії моделювання процесів. При проведенні моделювання інформаційних потоків та при розробленні методів використано теоретико-множинні підходи, алгебру систем, апарат модельно-орієнтованих підходів, методи концептуального моделювання, принципи побудови баз знань та формування логічного висновку, евристичні оцінки.

Наукова новизна отриманих результатів:

– набув подальшого розвитку метод ідентифікації залишкових дефектів у програмному забезпеченні, який відрізняється від відомих тим, що вхідна інформація про результати базового тестування обробляється штучною

нейронною мережею (ШНМ), і забезпечує визначення множини дефектів різного рівня критичності та аналіз цієї множини на наявність або відсутність залишкових дефектів;

– набула подальшого розвитку інформаційна технологія ідентифікації залишкових дефектів у програмних продуктах за рахунок опрацювання природомовних звітів про результати основного тестування, яка забезпечує висновок про відсутність або наявність залишкових дефектів у аналізованому програмному забезпеченні.

Практична значущість отриманих результатів полягає у проєктуванні та реалізації інтелектуальної інформаційної технології ідентифікації залишкових дефектів у програмних продуктах, яка забезпечує висновок про відсутність залишкових дефектів або про наявність залишкових дефектів з різними ступенями критичності їх наслідків на основі природомовного звіту про дефекти програмного продукту, виявлені під час його основного тестування.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

БД – база даних

ПП – програмний продукт

ПЗ – програмне забезпечення

ЖЦ – життєвий цикл

ШНМ – штучна нейронна мережа

МЗ – мобільні застосунки

ВСТУП

В процесі розробки програмного забезпечення (ПЗ) залишкові дефекти є суттєвою проблемою, яка може мати значні наслідки для якості та безпеки ПЗ. Залишкові дефекти – це дефекти, які залишаються невиявленими або невирішеними після його тестування та налагодження. Залишкові дефекти в програмних продуктах (ПП) загрожують їм відмовами або неправильною роботою. Це може призвести до часткової (аварійна зупинка) або повної зупинки (катастрофічна зупинка) роботи ПЗ, репутаційних втрат, інформаційних втрат, фінансових втрат і навіть людських жертв.

Отже, ці дефекти становлять загрозу для функціональності, продуктивності та надійності ПЗ. Тому, аналіз моделей та засобів виявлення залишкових дефектів має вирішальне значення для підвищення якості ПЗ та забезпечення його надійності.

Актуальність роботи полягає у виявленні дефектів, що залишилися в ПЗ після їх тестування, відповідно дослідження спрямоване на розробку інформаційної технології для виявлення дефектів, що залишилися в ПЗ.

Метою кваліфікаційної роботи є ідентифікація залишкових дефектів у ПЗ шляхом розробки методу та засобів інформаційної технології. Вивчаючи ці методи та засоби, можливо отримати уявлення про їх переваги та обмеження, а також зрозуміти їхню придатність для задач виявлення дефектів. Ці дослідження сприятимуть покращенню загального процесу розробки ПЗ.

Для досягнення поставленої мети потрібно розв'язати наступні взаємопов'язані *задачі:*

– проаналізувати існуючі методи та засоби ідентифікації залишкових дефектів у ПЗ;

– визначити напрямки поліпшення методів та засобів ідентифікації залишкових дефектів у ПЗ;

– розробити алгоритм ідентифікації залишкових дефектів у ПЗ;

– розробити метод ідентифікації залишкових дефектів у ПЗ;

– розробити вимоги до інформаційної технології ідентифікації залишкових дефектів у ПЗ;

– розробити структуру інформаційної технології ідентифікації залишкових дефектів у ПЗ;

– представити результати функціонування інформаційної технології ідентифікації залишкових дефектів у ПЗ.

Об'єктом дослідження є процес виявлення залишкових дефектів у ПЗ.

Предметом дослідження є методи та засоби інформаційної технології виявлення залишкових дефектів у ПЗ.

Наукова новизна отриманих результатів:

– набув подальшого розвитку метод ідентифікації залишкових дефектів у програмному забезпеченні, який відрізняється від відомих тим, що вхідна інформація про результати основного тестування обробляється штучною нейронною мережею (ШНМ), і забезпечує визначення множини дефектів різного рівня критичності та аналіз цієї множини на наявність або відсутність залишкових дефектів;

– набула подальшого розвитку інформаційна технологія ідентифікації залишкових дефектів у програмних продуктах за рахунок опрацювання природомовних звітів про результати основного тестування, яка забезпечує висновок про відсутність або наявність залишкових дефектів у аналізованому програмному забезпеченні.

Практична значущість отриманих результатів полягає у проєктуванні та реалізації інтелектуальної інформаційної технології ідентифікації залишкових дефектів у програмних продуктах, яка забезпечує висновок про відсутність залишкових дефектів або про наявність залишкових дефектів з різними ступенями критичності їх наслідків на основі природомовного звіту про дефекти програмного продукту, виявлені під час його базового тестування.

Методи дослідження. Для розв'язання поставлених задач використовуються основні положення загальної теорії систем, системного аналізу (ієрархічності, декомпозиції та інші), теорії моделювання процесів. При проведенні моделювання

інформаційних потоків та при розробленні методів використано теоретико-множинні підходи, алгебру систем, апарат модельно-орієнтованих підходів, методи концептуального моделювання, принципи побудови баз знань та формування логічного висновку, евристичні оцінки.

За темою кваліфікаційної роботи опубліковано одну статтю у матеріалах конференції, що індексуються в наукометричній базі Scopus, і одну статтю у фаховому виданні України [1, 2] (додаток А):

1) I. Zasornova, T. Hovorushchenko, M. Fedula, V. Buzyl. Intelligent Information Technology for Identification of Remaining Defects in Software Products // Proceedings of the 2023 IEEE 12-th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS-2023, Dortmund, 7-9 September 2023), vol. 1, pp. 33-37;

2) I. Zasornova, T. Hovorushchenko, O. Voichur. Study of Software Testing Tools According to the Testing Levels. Computer Systems & Information Technologies. 2023. №1. Pp. 38-46.

Крім цього, взято участь у The 12th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS). (September 7-9, 2023, Dortmund, Germany).

1 АНАЛІЗ ВІДОМИХ МОДЕЛЕЙ, МЕТОДІВ ТА ЗАСОБІВ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ

В останні роки використання ПП значно зросло практично у всіх предметних галузях (медицина, освіта, фінанси, торгівля, менеджмент, будівництво, сільське господарство, тощо) [3-7]. Кожен ПП повинен відповідати заданій специфікації, що гарантує якість та відповідність заданим вимогам замовника [8-10]. При цьому, тестування є необхідним процесом розробки ПП.

Залишкові дефекти відрізняються від інших типів дефектів тим, що вони виникають після завершення звичайних етапів тестування, таких як модульне або інтеграційне тестування. Ці приховані дефекти часто виникають у результаті складних взаємодій між компонентами системи або через непередбачувані сценарії користувача.

Виявлення та усунення цих залишкових дефектів є критично важливим, оскільки вони можуть негативно вплинути на стабільність, безпеку та зручність використання кінцевого продукту. Саме тому, є потреба у виявленні та знешкодженні дії цих дефектів.

Малі та середні ІТ-компанії часто мають проблеми з кваліфікацією робітників, які можуть бути задіяні для тестування ПП, що призводить до неможливості розв'язання поставлених задач тестування [11].

Навіть, якщо тестування ПП і проводиться в повному обсязі, воно не завжди є запорукою відсутності дефектів у ПП [12]. Дефекти залишаються у ПП внаслідок об'єктивних (недостатність коштів на тестування, неповнота тестів, недосконаліть методів тестування, тощо) та суб’єктивних (недостатня кваліфікація розробників та тестувальників ПЗ, вплив притаманних їм суб'єктивних недоліків, тощо) факторів [13-15].

Такі дефекти відносять до залишкових дефектів. Шкідливі результати прояву залишкових дефектів у розроблених ПП (наслідки, можливі матеріальні чи фінансові збитки, катастрофи, репутаційні втрати, людські втрати, тощо), їх наслідки та збитки, що виникли під час певної події, представлені в таблиці 1.1.

Таблиця 1.1 – Результати прояву залишкових дефектів у програмних продуктах

Подія	Наслідок	Збитки
1	2	3
Залишкові дефекти у автоматизованій системі для внутрішнього голосування компанії Shadow [16]	Неможливість проведення голосування демократичної партії США у 2020 році	Репутаційні втрати
Збої в інформаційних системах British Airways при онлайн-реєстрації та обслуговуванні рейсів у 2019 році [17]	Скасування 117 рейсів British Airways в аеропорті Heathrow, 10 рейсів British Airways в аеропорті Gatwick	200 млн дол. США
Залишкові дефекти в хронометражі автоматизованої системи першого космічного корабля CST-100 Starliner компанії Boeing у 2019 році [18]	Неможливість виходу корабля на необхідну орбіту і нестиковка із Міжнародною космічною станцією	Фінансові та репутаційні втрати
Неможливість стягнути борги за кредитами і втрата доходів компанії Provident Financial на суму 158 млрд дол. США у 2018 році [19] внаслідок прояву залишкових дефектів	Падіння вартості акцій британської фінансової компанії Provident Financial	2,2 млрд дол. США

Кінець таблиці 1.1

1	2	3
709 тис. листів з медичними даними не були доставлені пацієнтам або їх лікарям у Великій Британії в 2018 році через залишкові дефекти [19]	1700 випадків, коли вчасно неотримані дані призвели до серйозного погіршення стану здоров'я пацієнтів	Репутаційні втрати, людські втрати
Збільшення тюремних термінів ув’язнених в діапазоні від 0,5 до 1,5 року в США у 2018 році [19]	Сотні судових позовів в’язнів	Репутаційні втрати
Залишковий дефект в ПЗ вантажівок Fiat Chrysler у 2017 році, який призводив до відключення подушок та ременів безпеки [20]	Аварія з летальними наслідками, відкликання Fiat Chrysler 1,25 млн вже проданих вантажівок	Людські втрати, фінансові втрати, репутаційні втрати

Постійне збільшення кількості рядків програмного коду [21] та збільшення кількості новин про дефекти ПП [22], дають підстави зробити висновок про потенційне зростання кількості залишкових дефектів ПЗ. Відповідно до опублікованого звіту [23] компанії Undo спільно з Cambridge Business School (рисунок 1.1), компанії-розробники ПЗ витрачають на ідентифікацію дефектів (у тому числі й залишкових) 26% від усього часу на його виконання та 61 млрд дол. США з бюджету проєкту.

Загальна вартість проєктів, для акціонерів щорічно

Загальна кількість годин, витрачених на налагодження збоїв ПЗ щорічно

Загальна сума зарплати, витраченої на ідентифікацію дефектів щорічно

Рисунок 1.1 – Витрати часу та коштів на ідентифікацію дефектів [23]

1.1 Аналіз моделей ідентифікації залишкових дефектів у програмному забезпеченні

Для ідентифікації залишкових дефектів у ПЗ найбільш часто використовують чотири основні моделі:

– машинне навчання;

– статичний аналіз;

– динамічний аналіз;

– комбіновані.

Вибір моделі повинен базуватися на типі ПП, його об'ємі, бюджеті та інших факторах.

Моделі машинного навчання, що включають алгоритми глибоких нейронних мереж та дерев рішень показали свою перспективність у виявленні залишкових дефектів. Основою їх роботи є історично сформовані дані про дефекти, для розпізнавання патернів, які вказують на потенційні дефекти, які можуть бути присутні в нових випусках ПЗ.

Використовуючи різні механізми мережі, дослідники розробили нові моделі машинного навчання, призначені спеціально для виявлення дефектів. Моделі машинного навчання мають такі переваги, як здатність обробляти складні шаблони даних і можливість автоматизації [24, 25].

Проте, важливо враховувати обмеження, які можуть виникати з цим використанням моделей машинного навчання, до яких відносять великий обсяг навчальних даних і потенційні проблеми, пов'язані з великою кількістю інтерпретації.

Моделі статистичного аналізу надають ще один спосіб виявлення залишкових дефектів, використовуючи методи статистичного аналізу репозиторіїв коду [26]. Такі моделі пропонують альтернативний підхід до виявлення залишкових дефектів шляхом аналізу репозиторіїв коду. Вони використовують інструменти статистичного аналізу для отримання цінної інформації з великих наборів даних. Наприклад, автоматизоване виявлення та ідентифікація дефектів ПЗ у вбудованих системах транспортних засобів використовує статистичні моделі, такі як алгоритми кластеризації та класифікації. Хоча статистичні моделі доволі ефективні для виявлення прихованих шаблонів у репозиторіях коду, вони також можуть зіткнутися з труднощами, коли працюють зі складними або динамічними програмними системами [26].

Моделі динамічного аналізу ідентифікації залишкових дефектів у ПЗ спрямовані на виявлення та усунення залишкових дефектів під час виконання програми після пройдених етапів тестування та верифікації [27]. Динамічний аналіз коду може бути проведений під час розробки і забезпечує зворотній зв'язок у реальному часі, але залежать від точного моделювання поведінки системи [28]. Оцінка придатності моделей динамічного аналізу розробляємого ПЗ в основному залежить від наступних факторів: вимог до проєкту, наявності ресурсів, специфічних характеристик ПП, тощо. Моделі динамічного аналізу вимагають постійного моніторингу роботи програми під час її виконання. Це може бути здійснено за допомогою різних інструментів та технологій, які збирають дані про виконання програми і дозволяють аналізувати їх на предмет потенційних дефектів.

Аналіз дозволяє виявляти залишкові дефекти, такі як помилки у логіці програми, витоки пам'яті, невідповідне використання ресурсів та інші дефекти,

які можуть призвести до некоректної роботи програми або її аварійного завершення [29].

Для більш точного аналізу важливим є використання коректних даних під час тестування. Це дозволяє виявити можливі дефекти, пов'язані з обробкою даних, які програма отримує в реальному середовищі. Моделі динамічного аналізу також допомагають виявляти потенційні вразливості в безпеці програми, які можуть бути використані зловмисниками для атак. Це дозволяє підвищити безпеку розробляємого ПЗ [30, 31].

Для ефективного використання моделей динамічного аналізу для ідентифікації залишкових дефектів у ПЗ часто використовують засоби та системи автоматизації, які інтегруються в процес розробки та тестування продукту. Моделі динамічного аналізу можуть використовувати інструменти статичного аналізу для ретельної перевірки вихідного коду на наявність потенційних дефектів або вразливостей [28].

Перевагою моделей динамічного аналізу є їх здатність забезпечувати зворотний зв'язок у реальному часі під час процесу розробки; однак її ефективність значною мірою залежить від точного моделювання поведінки системи.

Також використання динамічної моделі є важливою частиною процесу контролю якості ПЗ і допомагає забезпечити стабільність, надійність та безпеку роботи ПЗ в реальних умовах використання. Найдосконалішими є моделі комбінованого аналізу, які можуть включати декілька попередньо описаних моделей.

Перераховані моделі, постійно удосконалюються та розробляються нові, які є перспективними для виявлення залишкових дефектів. Одна з таких розробок для виявлення дефектів ПЗ використовує навчання на малих вибірках [25].

Поєднання у процесах тестування нових моделей з існуючими може підвищити загальну ефективність виявлення залишкових дефектів. Серед найбільш відомих існуючих моделей на сьогодні широко використовують

наступні: McCall., Boehm, Dromey, FURPS, ISO 9126, ISO 25010. Характеристика перерахованих моделей наведена в додатку Б, таблиця Б.1.

Зазначені моделі спрямовані на поліпшення якості ПЗ і забезпечення його надійності та безпеки при роботі в реальному середовищі.

Проведений аналіз моделей для виявлення залишкових дефектів у ПЗ дозволив зробити висновок про їх важливість для підвищення якості та безпеки ПЗ. Завдяки використанню вище перерахованих основних моделей (машинне навчання; статичний аналіз; динамічний аналіз; комбінований аналіз) та розробці нових перспективних моделей, розробники можуть ефективно виявляти раніше невиявлені дефекти перед випуском ПЗ.

Лише розуміючи переваги та обмеження, пов'язані з кожною із моделей, команді проєкту необхідно приймати обґрунтоване рішення про те, які моделі найкраще відповідають їхнім потребам. Використання моделей може бути обмежено виділеними грошовими та апаратними ресурсами, які вкладені у процеси виявлення дефектів. Це може призвести до зниження надійності ПЗ та недостатньому задоволенню вимог користувачів.

1.2 Аналіз методів ідентифікації залишкових дефектів у програмному забезпеченні

Існуючі методи ідентифікації залишкових дефектів у ПЗ можна розділити за ступенем автоматизації на:

– неавтоматичні (без використання засобів автоматизації);

– напівавтоматичні (з частковим використанням засобів автоматизації);

– автоматичні (з використанням засобів автоматизації).

Також методи ідентифікації залишкових дефектів у ПЗ можна розділити на статичні та динамічні. Статичний метод аналізу вихідного коду передбачає його аналіз без виконання. При застосуванні такого методу можливо виявити потенційні проблеми, такі як виключення за нульовим вказівником або

переповнення буфера, досліджуючи властивості програми, такі як потік даних або потік керування.

Динамічний метод аналізу вихідного коду передбачає виконання тестових кейсів на працюючому ПЗ, щоб виявити дефекти під час виконання. Сюди входять такі методи, як модульне тестування, інтеграційне тестування та системне тестування.

Крім того, можна розділити існуючі методи на позитивні і негативні. Таке тестування проводять, щоб спостерігати за поведінкою ПЗ при його виконанні.

При проведенні негативного тестування, тобто введенні некоректних даних, можна знайти приховані дефекти або слабкі місця системи, які можуть призвести до виявлення залишкових дефектів.

Зазвичай негативне тестування виконують при перевірці полів і модулів: «Пошук», «Авторизація», «Реєстрація», «Зворотний зв'язок», «Форум», «Кошик», «Додати коментар» та інших. При позитивному тестуванні ті самі поля і модулі для введення інформації перевіряють введенням коректних даних.

Обов'язковим для перевірки є метод тестування продуктивності ПЗ, який виконують на завершальному етапі його розробки. Цей метод призначено для перевірки швидкості різних складових ПЗ таких як: відповідь сервера на запит, швидкість завантаження сторінок, швидкість завантаження зображень, здатність ПЗ витримувати певні види навантаження протягом певного періоду часу, швидкість обробки даних, роботу тегів різних видів та інших.

Кожен із методів має своє призначення, певні переваги та недоліки. Статичний метод аналізу вихідного коду дозволяє розробникам виявити потенційні проблеми на ранній стадії розробки, проте цим методом не можна виявити поведінку ПЗ, оскільки не передбачено його виконання. Динамічний метод аналізу вихідного коду дає цінну інформацію про те, як програма поводиться в реальних умовах, але такого аналізу може бути недостатньо через обмежене тестове покриття.

Аналіз методів ідентифікації залишкових дефектів у ПЗ тісно пов'язаний із життєвим циклом (ЖЦ) дефекта (рисунок 1.2).

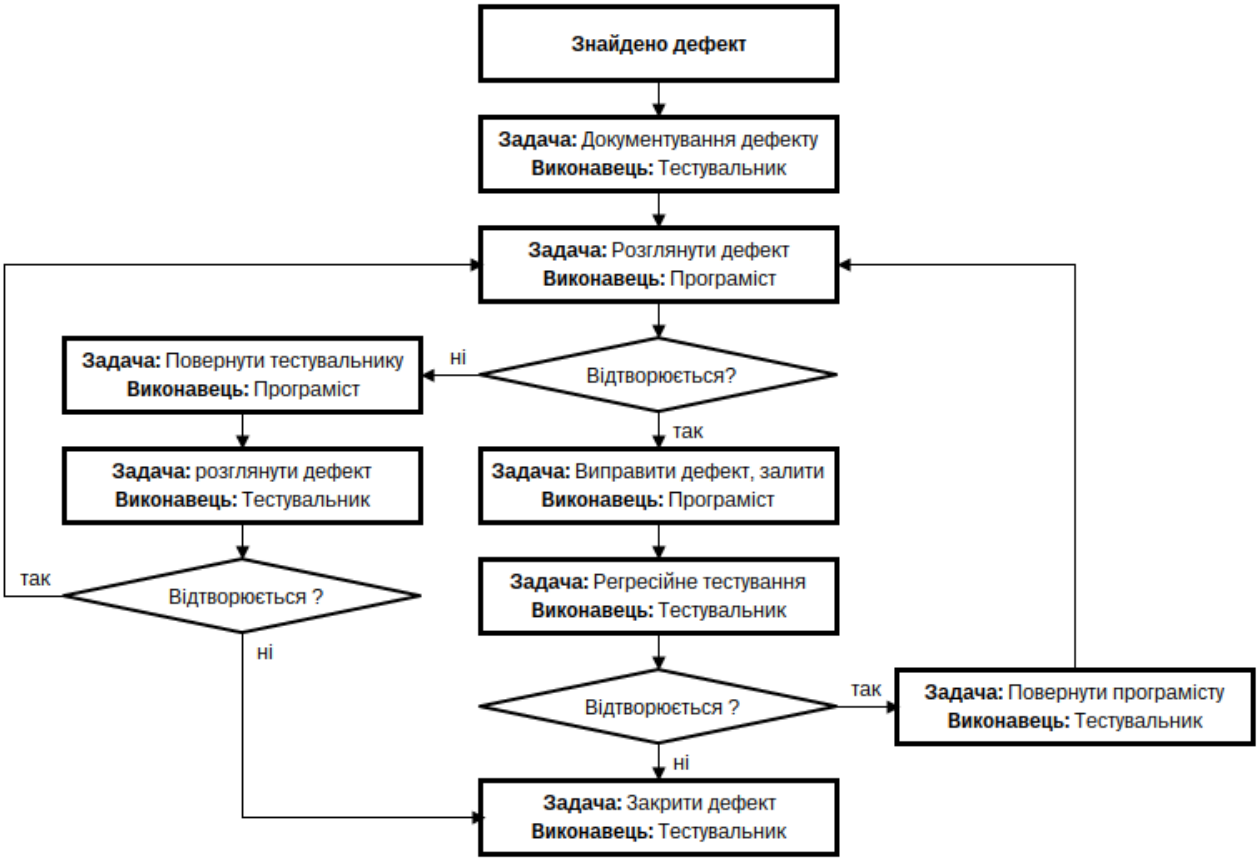


Рисунок 1.2 – Життєвий цикл дефекта

Під час життєвого циклу дефекта, отримані звіти у баг-трекінгових системах не завжди коректно відображають ефективність роботи розробників при виправленні дефектів. Така ситуація виникає тому, що дефекти різної пріоритетності зазвичай мають різну ступінь серйозності та складності, а баг-трекінгова система враховує лише поточний стан або статус дефекта. Наведений на рисунку 1.2 ЖЦ дефекту чітко розподіляє роботу програміста і тестувальника, що дозволяє сформувати послідовність використання різних методів ідентифікації залишкових дефектів. Для мінімізації роботи, компанії, які розробляють ПЗ, повинні використовувати наступну послідовність методів ідентифікації залишкових дефектів:

– статичне та динамічне рецензування коду (регулярне рецензування коду колегами може допомогти виявити та виправити залишкові дефекти на ранніх стадіях);

– автоматизоване тестування (впровадження автоматизованих тестових наборів, які охоплюють різні аспекти ПЗ, що дозволяє ефективно виявляти дефекти);

– безперервна інтеграція та розгортання (впровадження практик CI/CD забезпечує часте тестування та зворотний зв'язок, що дозволяє своєчасно виявляти дефекти);

– контроль версій (належний контроль версій полегшує відстеження змін, внесених до кодової бази, що дозволяє легше виявляти та виправляти дефекти);

– рефакторинг (реструктуризація коду для покращення його якості та зручності супроводу зменшує ймовірність внесення залишкових дефектів).

Кожен із перерахованих методів ідентифікації залишкових дефектів відіграє вирішальну роль у їх мінімізації. Усі методи повинні застосовуватись з урахуванням конкретних етапів розробки ПЗ.

Глибокий аналіз методів визначення залишкових дефектів ПЗ, з наданням цінної інформації виконано у роботі [32]. Авторами цієї праці досліджено вплив рефакторингу на взаємозв'язок між атрибутами якості та метриками дизайну, підкреслюючи, як певні модифікації можуть впливати на поширеність дефектів у ПЗ.

У працях [33, 34] проведені емпіричні дослідження оцінки серйозності помилок за допомогою метрик вихідного коду та методів статичного аналізу, аналізуючи способи точної оцінки серйозності залишкових дефектів.

Проведений аналіз методів ідентифікації залишкових дефектів у ПЗ дозволив зробити висновок про тісний взаємозв'язок обраних методів з ЖЦ дефекту.

Саме тому, для мінімізації роботи по ідентифікації залишкових дефектів необхідно використовувати: статичне та динамічне рецензування коду; автоматизоване тестування; безперервну інтеграцію та розгортання; контроль версій; рефакторинг.

1.3 Аналіз засобів ідентифікації залишкових дефектів у програмному забезпеченні

Існує багато різних засобів ідентифікації залишкових дефектів у ПЗ, серед яких є системи ідентифікації під час: автоматизованого тестування, моніторингу та журналювання, аналізу коду та інші.

Системи ідентифікації залишкових дефектів під час автоматизованого тестування є інструментами, що допомагають розробникам і тестувальникам автоматизувати процес тестування ПЗ. Такі системи дозволяють створювати, виконувати та керувати тестами автоматично, без необхідності ручного втручання. Вони надають можливість тестування на різних платформах, включаючи веб-, мобільні застосунки, настільні застосунки та інші.

Багато систем автоматизованого тестування підтримують різні мови програмування, що дозволяє розробникам здійснювати обґрунтований вибір. Вони інтегруються з іншими інструментами розробки та тестування, такими як: середовище розробки, системи контролю версій та системи збору зворотного зв'язку від користувачів. Також ці системи забезпечують створення детальних звітів про результати тестів та виявлені дефекти. Деякі із них дозволяють виконувати тести паралельно на декількох пристроях або у різних браузерах для прискорення тестування. Деякі автоматизовані системи можуть вимагати складних налаштувань і налагодження для досягнення бажаних результатів.

До переваг відомих засобів автоматизованого тестування можна віднести [35-38]: високу швидкість виконання тестових кейсів, відсутність використання ручного тестування, мінімізацію витрат та участі людини, можливість обробки великих обсягів даних [1].

До недоліків можна віднести [35-38]: потребу у висококваліфікованому персоналі та необхідність навчання персоналу для їх використання, високі витрати на їх розробку та впровадження, необхідність більш комплексного управління ризиками, непридатність при значних змінах вимог, швидке моральне

старіння при зміні технологічного прогресу, неможливість виявлення дефектів, що залишилися [1].

Системи моніторингу та журналювання дозволяють відстежувати та аналізувати роботу ПЗ в реальному часі. Вони фіксують події, дефекти, аномалії та відхилення в роботі програми, що допомагає ідентифікувати залишкові дефекти та забезпечувати високий рівень надійності та безпеки. Деякі системи можуть надавати графіки та аналітику для зручного аналізу даних.

Методи аналізу коду включають в себе статичний аналіз коду для виявлення потенційних дефектів та проблем у програмному коді. Вони можуть автоматично перевіряти код на відповідність стандартам програмування, безпекові уразливості та інші аспекти якості коду. Методи аналізу коду допомагають знижувати кількість дефектів, які можуть залишитися в ПЗ. Ці системи дозволяють збирати відгуки, скарги, побажання та іншу інформацію щодо використання ПЗ. Вони можуть включати в себе інструменти для збору коментарів, оцінок та інших даних від користувачів. Інформація, зібрана з цих систем, може бути використана для виявлення та виправлення залишкових дефектів, а також для вдосконалення функціональності та задоволення потреб користувачів. У роботі [2] авторами виконано порівняльний аналіз інструментів для автоматизованого тестування ПП (додаток В, таблиця В.1). У таблиці 1.2 наведено перелік найбільш використовуваних засобів ідентифікації дефектів та їх характеристика.

Таблиця 1.2 – Засоби ідентифікації залишкових дефектів у програмному забезпеченні

Засіб	Характеристика
1	2
Системи автоматизованого тестування	
Selenium	Фреймворк для автоматизованого тестування веб-застосунків, що дозволяє створювати та виконувати тести на веб-сайтах та застосунках. Підтримує різні мови програмування та браузери

Продовження таблиці 1.2

1	2
Appium	Відкритий інструмент для автоматизованого тестування мобільних застосунків на платформах iOS та Android. Він дозволяє розробникам та тестувальникам створювати тести для мобільних застосунків на різних пристроях
JUnit	Фреймворк для автоматизованого тестування для Java, який дозволяє створювати та виконувати тести для Java-програм. Він надає інструменти для написання та виконання тестів одиниць коду
TestNG	Фреймворк для автоматизованого тестування, який спеціалізується на розширених можливостях тестування для Java-програм. Він підтримує розподілені тести, параметризовані тести та інші функції
Cucumber	Інструмент для автоматизованого тестування, який базується на використанні природної мови для опису тестів. Він дозволяє бізнес-аналітикам та розробникам створювати тести, які можуть бути прочитані і розуміються усіма учасниками проєкту
TestComplete	Інструмент для автоматизованого функціонального тестування, який підтримує автоматизацію на різних платформах, включаючи веб-, мобільні та настільні застосунки
Robot Framework	Фреймворк для автоматизованого тестування, який дозволяє створювати тести за допомогою ключових слів і природних мовних конструкцій. Він підтримує тести для різних типів ПЗ
Apache JMeter	Інструмент для проведення навантажувальних тестів та тестів продуктивності. Він дозволяє моделювати навантаження на веб-сервери, сервіси та застосунки для оцінки їхньої продуктивності та надійності

Продовження таблиці 1.2

1	2
Postman	Інструмент для тестування API, який надає інтерфейс для створення, відправки та аналізу запитів до веб-сервісів. Він дозволяє тестувальникам та розробникам перевіряти функціональність API та взаємодіяти з ними
SoapUI	Інструмент для тестування веб-сервісів та SOAP-протоколу. Він дозволяє створювати та виконувати тести для веб-сервісів, перевіряти їхню функціональність та взаємодіяти з ними
Моніторинг та журналювання	
Splunk	Платформа для аналізу журналів та моніторингу великих обсягів даних. Вона використовується для збору, аналізу та візуалізації лог-файлів та даних з різних джерел для виявлення аномалій та проблем
ELK Stack (Elasticsearch, Logstash, Kibana)	Набір інструментів, які дозволяють збирати, обробляти та візуалізувати дані журналів для виявлення аномалій та проблем
New Relic	Інструмент для моніторингу продуктивності та доступності веб-застосунків
Nagios	Система моніторингу мережі та сервісів, яка дозволяє відстежувати стан обладнання та ПЗ
Prometheus	Відкрите ПЗ для моніторингу та аналізу метрик великих систем
Grafana	Інструмент для візуалізації та аналізу даних, який може інтегруватися з іншими системами моніторингу, такими як Prometheus
Loggly	Хмарний сервіс для аналізу лог-файлів та журналів
Graylog	Відкрите ПЗ для аналізу журналів та журналювання.

Продовження таблиці 1.2

1	2
	Воно надає інструменти для збору та аналізу лог-файлів, а також можливості пошуку та фільтрації даних
Datadog	Сервіс моніторингу та аналізу продуктивності, який надає інформацію про стан систем, мереж та застосунків
Log4j	Бібліотека для журналювання у Java- застосунках. Вона дозволяє реєструвати події та дії в програмному коді для аналізу та виявлення проблем у програмі
Методи аналізу коду	
IntelliJ IDEA, Visual Studio, PyCharm	Інтегровані середовища розробки (IDE) зі вбудованими інструментами аналізу, які допомагають розробникам виявляти та виправляти потенційні дефекти в програмному коді
SonarQube, PMD, FindBugs, ESLint, Pylint	Статичні аналізатори коду, які автоматично перевіряють програмний код на наявність дефектів, стандартів програмування та безпекових уразливостей
Checkmarx, Fortify, OWASP ZAP	Аналізатори уразливостей, спеціалізовані на виявленні безпекових уразливостей та дефектів у програмному коді, які можуть призвести до потенційних загроз безпеці
Інші засоби	
Ручне тестування та баг-репорти	Ручне тестування включає в себе процеси, під час яких тестувальники вручну перевіряють функціональність та взаємодію з ПЗ та складають звіти про знайдені дефекти
Автоматизовані системи збору зворотного зв'язку	Використовуючи засоби збору зворотного зв'язку з веб-сайтів та застосунків, системи дозволяють збирати відгуки, скарги, побажання та іншу інформацію від користувачів щодо використання ПЗ
Google	Аналітичні інструменти дозволяють збирати та аналізувати

Кінець таблиці 1.2

1	2
Analytics, Mixpanel	дані про поведінку користувачів, використовуючи веб-сайти та застосунки, що допомагає розробникам покращувати функціональність та вдосконалювати досвід користувачів

Отже, розглянуті засоби (системи автоматизованого тестування, моніторинг та журналювання, методи аналізу коду та інші) можливо використовувати при ідентифікації залишкових дефектів у ПЗ на різних стадіях розробки ПП. В роботі пропонується використовувати автоматизовані засоби тестування.

1.4 Висновки. Постановка задачі

На основі проведеного аналізу можна розробити метод та засоби інформаційної технології ідентифікації залишкових дефектів у ПЗ. Застосовуючи їх можливо ефективно мінімізувати появу залишкових дефектів під час розробки ПЗ. Аналіз дозволив виявити, що основні моделі (машинне навчання; статичний аналіз; динамічний аналіз; комбінований аналіз) та нові перспективні моделі можуть ефективно виявляти раніше невиявлені дефекти перед випуском ПЗ.

Проведений аналіз методів ідентифікації залишкових дефектів у ПЗ дозволив виявити тісний взаємозв'язок обраних методів з ЖЦ дефекту. Це дозволило виявити методи для ідентифікації залишкових дефектів а саме: статичне та динамічне рецензування коду; автоматизоване тестування; безперервну інтеграцію та розгортання; контроль версій; рефакторинг.

Окрім того, виявлено, що існуючі засоби ідентифікації дефектів можуть бути використані у процесі розробки та тестування і охоплювати весь ЖЦ ПЗ.

Основними напрямками покращення є: впровадження стратегії автоматизованого тестування в поєднанні з ретельними перевірками коду; покращення ефективності у виявленні та усуненні дефектів; інтегрування безперервної інтеграції та розгортання у робочий процес розробки.

Наведені напрямки покращення методу та засобів ідентифікації залишкових дефектів у ПЗ можуть призвести до покращення якості ПЗ шляхом зменшення часу на виявлення залишкових дефектів та їх кількості.

Проведений аналіз також виявив, що при впровадженні нових методів ідентифікації залишкових дефектів можуть виникнути проблеми, пов'язані з: опором змінам з боку розробників; недостатньою кваліфікацією тестувальників та програмістів; обмеженістю ресурсів для тестування та розробки ПЗ. Подолання цих проблем вимагає навчання персоналу та виділенні достатніх ресурсів на розробку ПЗ і його тестування.

Подальші напрямки досліджень можуть включати: створення передових методів, таких як ідентифікації дефектів на основі штучного інтелекту; вивчення впливу конкретних методів створення ПП на виникнення залишкових дефектів.

Аналіз показав, що виявлення та виправлення залишкових дефектів має вирішальне значення для забезпечення високої якості ПЗ. Використовуючи розглянуті методи та засоби, компанії можуть ефективно мінімізувати витрати на розробку і зменшити кількість залишкових дефектів. Для цього у кваліфікаційній роботі необхідно вирішити такі взаємопов'язані задачі:

– проаналізувати існуючі методи та засоби ідентифікації залишкових дефектів у ПЗ;

– визначити напрямки поліпшення методів та засобів ідентифікації залишкових дефектів у ПЗ;

– розробити алгоритм ідентифікації залишкових дефектів у ПЗ;

– розробити метод ідентифікації залишкових дефектів у ПЗ;

– розробити вимоги до інформаційної технології ідентифікації залишкових дефектів у ПЗ;

– розробити структуру інформаційної технології ідентифікації залишкових дефектів у ПЗ;

– представити результати функціонування інформаційної технології ідентифікації залишкових дефектів у ПЗ.

2 ІНФОРМАЦІЙНІ ПОТОКИ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ІДЕНТИФІКАЦІЇ ЗАЛИШКОВИХ ДЕФЕКТІВ У ПРОГРАМНОМУ ЗАБЕЗПЕЧЕННІ

2.1 Концепція інформаційної технології ідентифікації залишкових дефектів у програмному забезпеченні

Відомо, що навіть після тестування, в ПЗ зазвичай залишаються дефекти через об'єктивні (некоректні процедури тестування та неповні тести, недостатнє фінансування, тощо) та суб'єктивні (недостатня підготовка розробників ПЗ та тестувальників, вплив їх суб'єктивних недоліків, тощо) фактори [16, 17, 39–41].

Коли ПЗ тестується і налагоджується, залишкові дефекти існують (у певний момент) і є ще не розпізнаними, що відрізняє їх від виявлених дефектів. На графіку (рисунок 2.1) та у таблиці 2.1 відображено типову щільність дефектів на 1000 рядків коду для ПЗ різного розміру [39, 42-48].

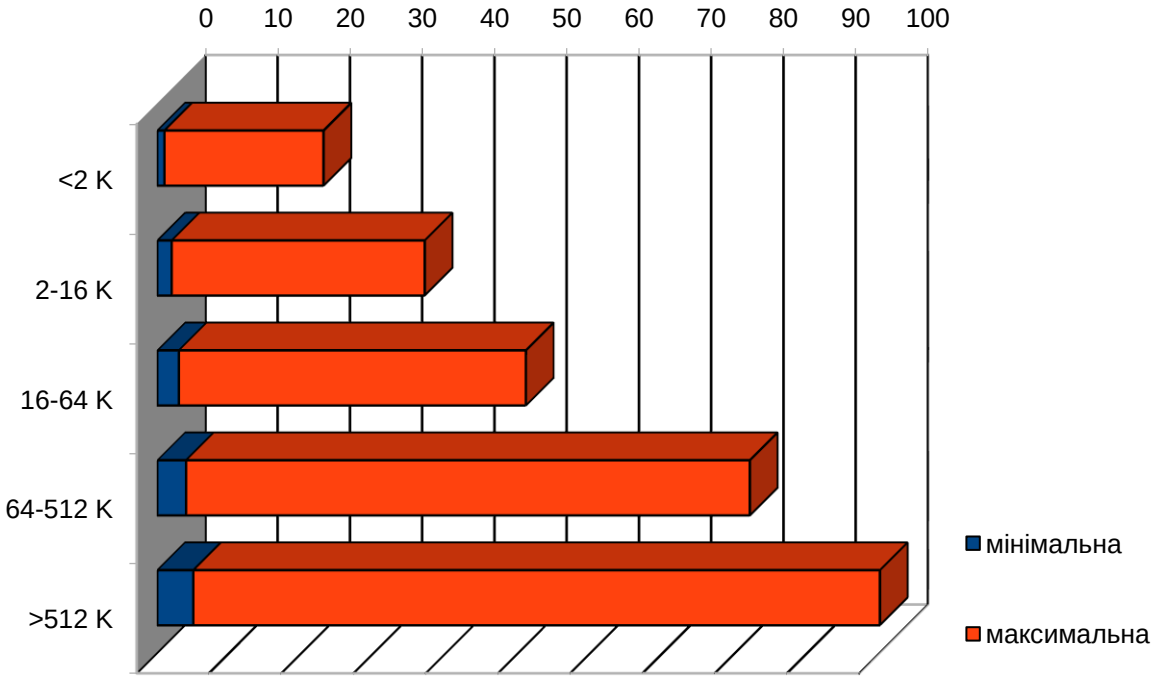


Рисунок 2.1 – Діаграма типової щільності дефектів на 1000 рядків коду для ПЗ різного розміру

Таблиця 2.1 – Типова щільність дефектів на 1000 рядків коду для ПЗ різного розміру

Щільність дефектів	Кількість рядків коду				
	Менше 2 тисяч	2-16 тисяч	16-64 тисяч	64-512 тисяч	Більше 512 тисяч
мінімальна	1	2	3	4	5
максимальна	22	35	48	78	98

З таблиці 2.1 чітко видно зростання щільності дефектів в залежності від кількості рядків програмного коду. Так, мінімальна щільність дефектів при кількості коду менше двох тисяч складає – 1, а при більше 512 тисяч – 5. Відповідно, максимальна щільність дефектів при кількості коду менше двох тисяч складає – 22, а при більше 512 тисяч – 98.

Розробники повинні переконатися, що ПП відповідає необхідним стандартам якості та надійності, пам'ятаючи при цьому, що може залишитися невелика кількість дефектів, які необхідно знайти, виправити або мінімізувати до рівня, коли їх вплив буде керованим.

Для розробки концепції інформаційної технології ідентифікації залишкових дефектів у ПЗ необхідно виконати класифікацію залишкових дефектів за ступенем критичності їх наслідків. Усі існуючі дефекти за ступенем критичності їх наслідків можна розділити на: незначні, помірні, серйозні, катастрофічні.

Розглянемо більш детально кожну ступінь критичності дефектів:

– незначні (ступінь критичності їх наслідків становить 1) – дефекти ПП, за наявності яких ПП є придатним для використання без втрати функційності;

– помірні (ступінь критичності їх наслідків становить 2) – дефекти ПП за наявності яких ПП є придатним для використання, проте з частковою втратою функційності;

– серйозні (ступінь критичності їх наслідків становить 3) – дефекти ПП, за наявності яких ПП непридатний для використання через генерування хибних результатів;

– катастрофічні (ступінь критичності їх наслідків становить 4) – дефекти ПП, за наявності яких ПП непридатний для використання через спотворення даних та відмови роботи ПЗ.

Як було доведено у [39], накопичення дефектів з певним, нижчим, ступенем критичності їх наслідків можуть бути причиною виникнення залишкових дефектів з наступними, вищими, ступенями критичності їх наслідків. Накопичення певної кількості незначних дефектів призводить до появи помірних дефектів, накопичення певної кількості яких, в свою чергу, призводить до появи серйозних дефектів, накопичення певної кількості яких, відповідно, призводить до появи катастрофічних дефектів. Це обґрунтовується тим фактом, що чим довше дефект перебуває у циклі розроблення ПЗ, тим сильніше він проникає в усі компоненти ПЗ і тим більше шкоди завдає на всіх етапах на всі компоненти. Отже, дефекти одного рівня критичності можуть бути причиною виникнення залишкових дефектів не лише наступного рівня критичності, але й залишкових дефектів вищих рівнів критичності. В такому разі введемо поріг допустимої кількості дефектів, при перевищенні якого необхідно приймати висновок про наявність залишкових дефектів наступного рівня критичності.

Початковими даними для формування висновку про наявність чи відсутність залишкових дефектів певного рівня критичності є інформація про кількість і типи дефектів, виявлених під час базового тестування, тобто звіт базового тестування. Знаходження чим більшої кількості залишкових дефектів підвищить достовірність процесу тестування і, відповідно, якість ПЗ, що, у свою чергу, уможливить розроблення бездефектного ПЗ.

Отже, для ідентифікації у ПП залишкових дефектів з різними ступенями критичності їх наслідків необхідна інформація про дефекти (їх кількість і типи) ПП, які були виявлені під час його тестування. На основі опрацювання такої природомовної інформації нам потрібно отримати висновок про відсутність залишкових дефектів або про їх наявність з різними ступенями критичності їх наслідків. Таку задачу важко формалізувати через складність опрацювання

початкових даних про наявні дефекти. Окрім того, потрібно врахувати взаємний вплив виявлених та залишкових дефектів ПП та інші фактори.

Тому, для розв'язання такої задачі необхідно використати штучну нейронну мережу (ШНМ).

Запропонована концепція ідентифікації залишкових дефектів у ПП представлена на рисунку 2.2.

Рисунок 2.2 – Концепція ідентифікації залишкових дефектів у ПП

Отже, концепцією, яка є основою інтелектуальної інформаційної технології ідентифікації залишкових дефектів у ПП є ідентифікація залишкових дефектів з різними ступенями критичності їх наслідків на основі інформації про дефекти ПП, виявлені під час його тестування з використанням ШНМ.

2.2 Препроцесінг та опрацювання даних для ідентифікації залишкових дефектів у програмному забезпеченні

Для опису правил формування висновку про встановлення наявності залишкових дефектів введемо поріг $r_i \in R$ ($R = \{r_j \mid j = 1..k\}$, де r_j – поріг допустимої кількості дефектів, при перевищенні якого формується висновок про наявність залишкових дефектів, j – кількість типів порогів, що змінюється від 1 до k, k – кількість рівнів критичності дефектів (тобто наразі $k = 4$)).

Загальне правило. Якщо відношення сумарного значення дефектів i-го рівня критичності до загальної кількості виявлених під час базового тестування дефектів (або сумарне значення дефектів i-го рівня критичності) перевищує поріг r_i, то приймається висновок про наявність залишкових дефектів $(i+1)$-го (наступного) рівня критичності.

Із загального правила витікають наступні *конкретні правила формування висновку про наявність залишкових дефектів*:

1) Якщо відношення сумарного значення дефектів 1-го рівня критичності (незначних дефектів) до загальної кількості виявлених під час базового тестування дефектів дорівнює або перевищує поріг r_1, то приймається висновок про наявність залишкових дефектів 2-го рівня критичності (помірних дефектів);

2) Якщо відношення сумарного значення дефектів 2-го рівня критичності (помірних дефектів) до загальної кількості виявлених під час базового тестування дефектів дорівнює або перевищує поріг r_2, то приймається висновок про наявність залишкових дефектів 3-го рівня критичності (серйозних дефектів);

3) Якщо сумарне значення дефектів 3-го рівня критичності (серйозних дефектів) перевищує поріг r_3, то приймається висновок про наявність залишкових дефектів 4-го рівня критичності (катастрофічних дефектів);

4) Якщо сумарне значення дефектів 4-го рівня критичності (катастрофічних дефектів) перевищує поріг r_4, то приймається висновок про непридатність ПЗ і можливу відмову системи. r_i

Уточнення розподілу типу дефектів ПЗ для формування висновку про наявність залишкових дефектів ПЗ проведено в цьому дослідженні. Тому, порогові значення кількості дефектів кожного рівня критичності, по перевищенню яких приймається висновок (про наявність або відсутність залишкових дефектів із зазначенням рівня(ів) критичності залишкових дефектів), не є описаними у відомих літературних джерелах. Для встановлення цих порогових значень проводились дослідження кількості дефектів ПЗ. Було досліджено ПЗ, яке розроблене софтверними компаніями м. Хмельницького. Воно вміщувало різну кількість рядків коду, з врахуванням і без врахування впливу дефектів одного рівня критичності на виникнення дефектів наступного рівня критичності. Результати дослідження наведені в таблиці 2.2.

Таблиця 2.2 – Кількість дефектів ПЗ з урахуванням і без урахування впливу дефектів одного рівня критичності на виникнення дефектів наступного рівня критичності

Параметри	Кількість дефектів ПЗ					
	без урахування			з урахуванням		
	впливу дефектів одного рівня критичності на виникнення дефектів наступного рівня критичності					
Кількість рядків коду	100	1000	10000	100	1000	10000
Незначні дефекти	8	19	51	8	19	51
Помірні дефекти	2	5	16	5	14	33
Серйозні дефекти	0	1	1	2	2	1
Катастрофічні дефекти	0	0	0	1	1	0
Всього	10	25	68	16	36	85

В результаті проведеного дослідження (таблиця 2.2) можна зробити наступні висновки:

– для програмного коду зі 100 операторів кількість незначних дефектів становить (8/10=0.8) 0.8 (перевищує 0.75) загальної кількості виявлених без

врахування взаємовпливу дефектів одного рівня критичності на виникнення дефектів наступного рівня критичності, і через це виникли (5–2=3) три помірні дефекти; кількість помірних дефектів складає (5/10=0.5) 0.5 загальної кількості виявлених без врахування взаємовпливу дефектів одного рівня критичності на виникнення дефектів наступного рівня критичності, і через це виникли (2–0=2) два серйозні дефекти, які спричинили появу (1–0=1) одного катастрофічного дефекту;

– для програмного коду з 1000 операторів кількість незначних дефектів становить (19/25=0.76) 0.76 (перевищує 0.75) загальної кількості виявлених без врахування взаємовпливу дефектів одного рівня критичності на виникнення дефектів наступного рівня критичності, і через це виникли (14–5=9) дев'ять помірних дефектів; кількість помірних дефектів складає (14/25=0.56) 0.56 (перевищує 0.5) загальної кількості виявлених без врахування взаємовпливу дефектів одного рівня критичності на виникнення дефектів наступного рівня критичності, і через це виник один серйозний дефект, два серйозні дефекти спричинили появу одного катастрофічного дефекту;

– для програмного коду з 10000 операторів кількість незначних дефектів становить (51/68=0.75) 0.75 загальної кількості виявлених без врахування взаємовпливу дефектів одного рівня критичності на виникнення дефектів наступного рівня критичності, і через це виникли (33–16=17) 17 помірних дефектів; кількість помірних дефектів складає (33/68=0.49) 0.49 (не перевищує 0.5) загальної кількості виявлених без врахування взаємовпливу дефектів одного рівня критичності на виникнення дефектів наступного рівня критичності, тому серйозні дефекти з причини накопичення помірних дефектів не виникли; один же серйозний дефект не спричинив появу катастрофічного дефекту.

В такому разі порогові значення кількостей дефектів кожного рівня критичності, по перевищенню яких приймається висновок про встановлення наявності або відсутності залишкових дефектів (із зазначенням рівня(ів) критичності залишкових дефектів у разі встановлення їх наявності), вводяться на основі евристичних оцінок (за таблицею 2.2) наступним чином:

– якщо відношення сумарного значення дефектів 1-го рівня критичності (незначних дефектів) до загальної кількості виявлених під час базового тестування дефектів dn_1 дорівнює або перевищує 0.75, то приймається висновок про встановлення наявності залишкових дефектів 2-го рівня критичності (помірних дефектів);

– якщо відношення сумарного значення дефектів 2-го рівня критичності (помірних дефектів) до загальної кількості виявлених під час базового тестування дефектів dn_2 дорівнює або перевищує 0.5, то приймається висновок про встановлення наявності залишкових дефектів 3-го рівня критичності (серйозних дефектів);

– якщо кількість дефектів dn_3 3-го рівня критичності (серйозних) перевищує 2, то приймається висновок про встановлення наявності залишкових дефектів 4-го рівня критичності (катастрофічних дефектів);

– якщо кількість дефектів dn_4 4-го рівня критичності (катастрофічних) перевищує або дорівнює 1, то приймається висновок про непридатність ПЗ і можливу відмову системи.

Отже, пороги мають наступні значення: r_1=0.75; r_2=0.5; r_3=2; r_1=1.

При аналізі таблиці 2.2 видно, що при перевищенні саме таких значень виникають залишкові дефекти більш високих рівнів критичності, тому саме ці значення використовуються в якості порогових при формуванні висновку про встановлення наявності або відсутності залишкових дефектів (із зазначенням рівня(ів) критичності залишкових дефектів у разі встановлення їх наявності).

Для полегшення сприйняття отриманих даних (таблиця 2.2) можна здійснити перерахування у відсотки відносно загальної кількості дефектів ПЗ (з урахуванням кількості рядків коду).

Використовуючи перераховані дані була побудована діаграма відсоткового відношення кількості дефектів ПЗ з урахуванням і без урахування впливу дефектів одного рівня критичності на виникнення дефектів наступного рівня критичності до загальної кількості дефектів (рисунок 2.3).

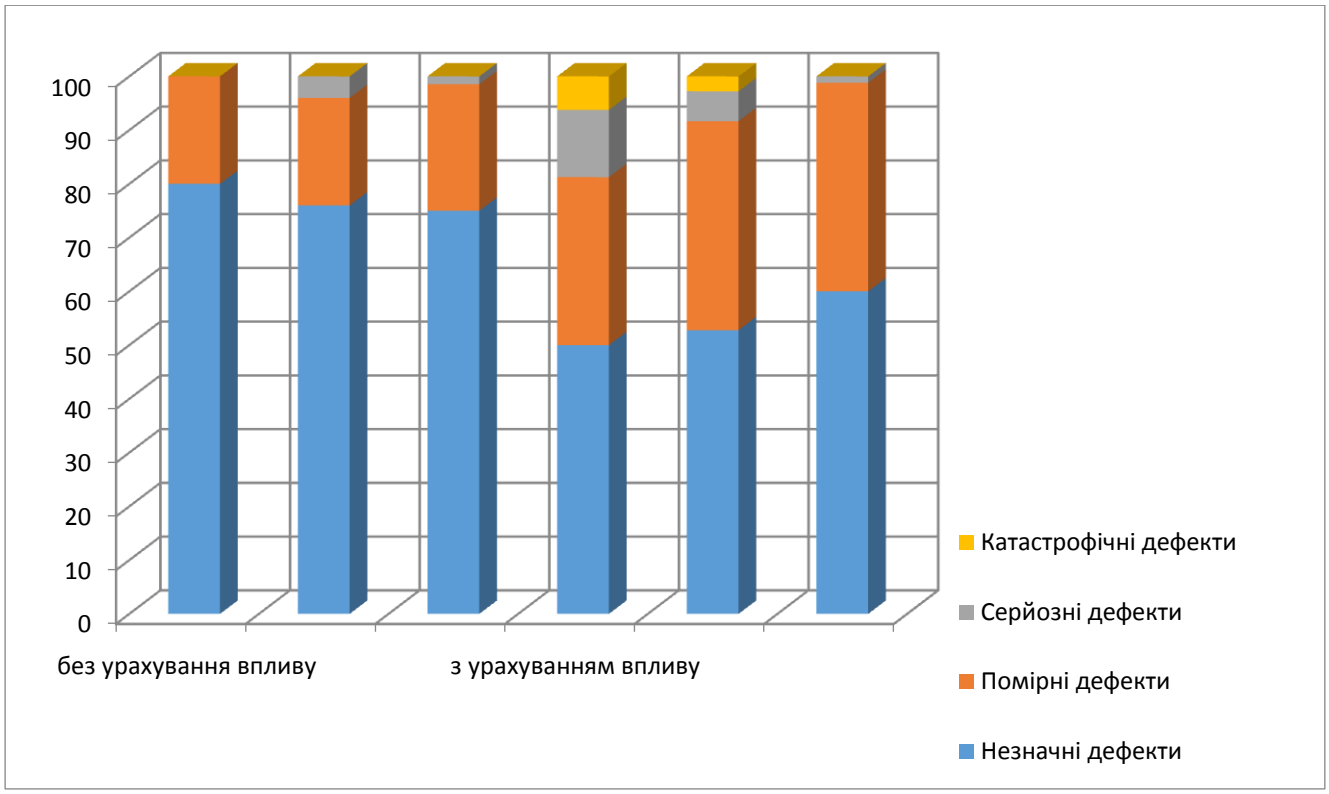


Рисунок 2.3 – Діаграма відсоткового відношення кількості дефектів ПЗ з урахуванням і без урахування впливу дефектів одного рівня критичності на виникнення дефектів наступного рівня критичності до загальної кількості дефектів

З діаграми (рисунок 2.3) видно, як збільшується кількість дефектів ПЗ з урахуванням впливу дефектів одного рівня критичності на виникнення дефектів наступного рівня критичності. Отже, при перевищенні порогових значень виникають залишкові дефекти більш високих рівнів критичності.

2.3 Інформаційні потоки для ідентифікації залишкових дефектів у програмному забезпеченні

Основними джерелами інформації для встановлення наявності/відсутності залишкових дефектів є природомовна інформація про кількість і типи дефектів, виявлених під час основного тестування, тобто звіт основного тестування, що описує дефекти, виявлені під час тестування.

Інформаційні потоки для ідентифікації залишкових дефектів у ПЗ представляють собою систему обміну даними та інформацією, спрямовану на виявлення та аналіз потенційних дефектів і недоліків в ПП після завершення фази розробки та тестування. Ці інформаційні потоки грають важливу роль у процесі забезпечення якості ПЗ та підвищенні надійності продукту. Основні етапи та складові інформаційних потоків для ідентифікації залишкових дефектів у ПЗ включають: збір даних, моніторинг та аналіз, повідомлення про дефекти, виправлення та тестування, повторний моніторинг, систематизація інформації, верифікація та валідація, звітність та аналіз результатів. На етапі збору даних здійснюється збір інформації про роботу ПЗ в реальних умовах експлуатації. Інформація може бути зібрана з різних джерел, таких як журнали подій, звіти про дефекти, звернення користувачів та інші. Для виявлення можливих дефектів і проблем у роботі ПЗ зібрані дані піддають моніторингу та аналізу. Аналіз може включати в себе статистичні методи, а також використання алгоритмів машинного навчання для виявлення аномалій та виявлення закономірностей в даних. Виявлені дефекти і проблеми фіксують та документують, іноді навіть класифікують за серйозністю та пріоритетом. Ця інформація передається розробникам та тестувальникам для подальшого аналізу та виправлення. Розробники вносять необхідні зміни в програмний код для усунення дефектів, після чого проводяться тестування для підтвердження виправлень і визначення ефективності виправлень. Після внесення змін та виправлень у ПП, інформаційні потоки знову піддаються моніторингу для перевірки ефективності виправлень та виявлення можливих нових дефектів. Процес опрацювання інформаційних потоків для ідентифікації залишкових дефектів у ПЗ сприяє покращенню якості ПП і забезпеченню його надійності в реальних умовах експлуатації. Всі виявлені дефекти та зміни документують у системах управління дефектами. Це дозволяє зберігати всю інформацію про дефекти та виправлення в структурованому та доступному для аналізу вигляді. Після виправлень та тестування ПЗ повинно бути піддане процесу верифікації та валідації, щоб переконатися, що всі дефекти були успішно усунуті та продукт відповідає специфікаціям та вимогам. Далі

генеруються звіти та аналізи результатів. Це дозволяє команді розробників та керівництву отримати інформацію про ефективність процесу та приймати рішення щодо подальших кроків у розвитку ПП. Процес ідентифікації залишкових дефектів є циклічним, і його проведення повторюється після кожної нової версії ПП. Це допомагає постійно вдосконалювати якість ПЗ та забезпечувати користувачам надійну роботу продукту.

Загалом, інформаційні потоки для ідентифікації залишкових дефектів у ПЗ (рисунок 2.3) є невід'ємною частиною процесу забезпечення якості ПП та підтримки його роботи в реальних умовах. Вони допомагають виявляти та виправляти дефекти, забезпечуючи високий рівень надійності та ефективності програми.

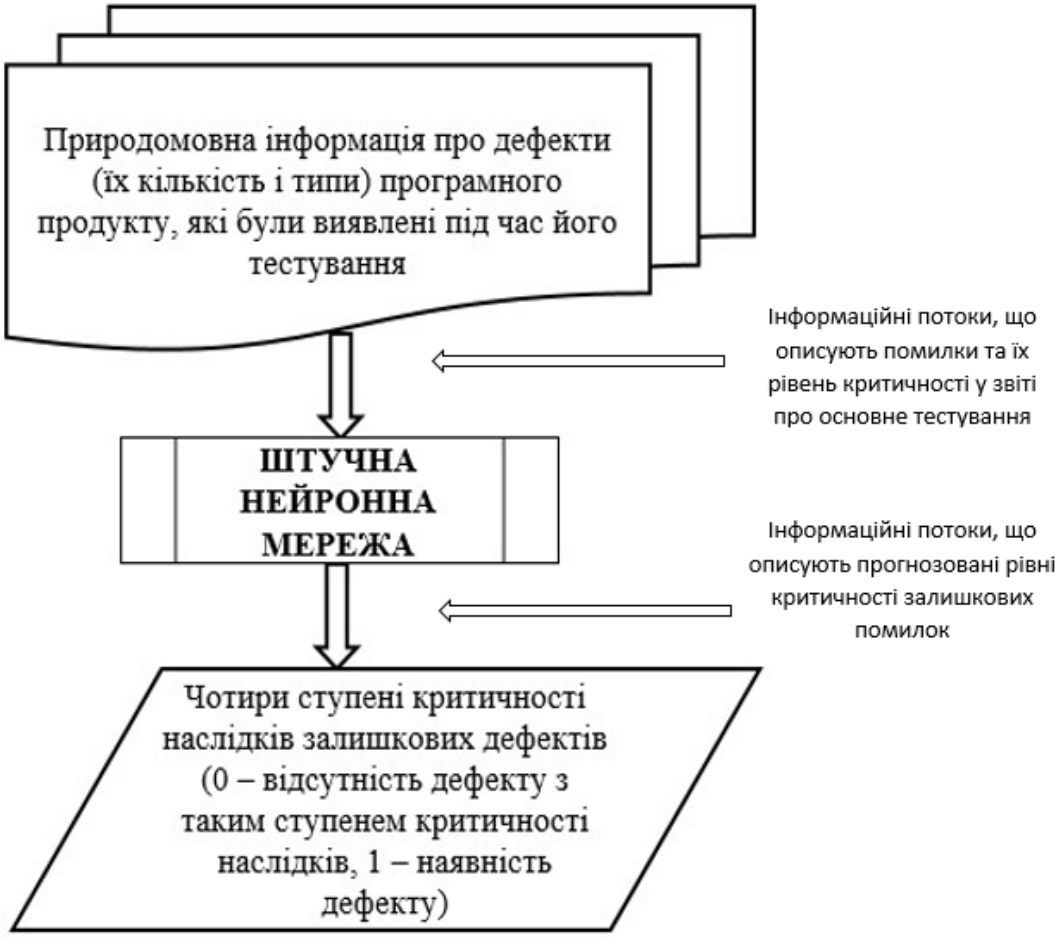


Рисунок 2.4 – Структура та зміст інформаційних потоків при ідентифікації залишкових дефектів у програмному забезпеченні

Також важливо підтримувати взаємодію з користувачами ПП для отримання повідомлень про можливі проблеми та дефекти, які вони виявили в процесі використання. Запити, відгуки та повідомлення від користувачів є важливим джерелом інформації для ідентифікації залишкових дефектів.

Для деяких ПП, таких як вбудовані системи, IoT-пристрої або ПЗ для медичних пристроїв, ідентифікація залишкових дефектів може вимагати тестування на реальних об'єктах або в реальних умовах. Це може включати в себе відслідковування працездатності та надійності пристроїв у реальних сценаріях використання.

Великі програмні проєкти можуть використовувати автоматизовані системи для ідентифікації залишкових дефектів. Це включає в себе автоматизоване тестування, моніторинг і аналіз логів, які дозволяють виявляти аномалії та дефекти в реальному часі.

Загальна мета інформаційних потоків для ідентифікації залишкових дефектів – забезпечити найвищу якість та надійність ПЗ, видаливши всі можливі дефекти, які можуть виникнути під час його використання. Цей процес є невід'ємною частиною ЖЦ розробки ПП та вимагає постійного вдосконалення та оптимізації для забезпечення задоволеності користувачів і успішної експлуатації продукту на ринку.

2.4 Висновки

У другому розділі доведено, що зростання щільності дефектів залежить від кількості рядків програмного коду. Визначено типову щільність дефектів на 1000 рядків коду для ПЗ різного розміру. Так, мінімальна щільність дефектів при кількості коду менше двох тисяч складає – 1, а при більше 512 тисяч – 5. Відповідно, максимальна щільність дефектів при кількості коду менше двох тисяч складає – 22, а при більше 512 тисяч – 98.

З метою ідентифікації у ПЗ залишкових дефектів з різними ступенями критичності їх наслідків розглянуто інформацію про дефекти, а саме про їх кількість і типи, які були виявлені під час його тестування. Для отримання висновку про відсутність або наявність залишкових дефектів з різними ступенями критичності їх наслідків, в роботі рекомендовано враховувати взаємний вплив виявлених та залишкових дефектів ПЗ та інші фактори. Для розв'язання такої задачі запропоновано використовувати ШНМ. Для цього розроблено концепцію інформаційної технології ідентифікації залишкових дефектів у ПЗ.

Доведено, якщо відношення сумарного значення дефектів 1-го рівня критичності (незначних дефектів) до загальної кількості виявлених під час базового тестування дефектів дорівнює або перевищує 0.75, то приймається висновок про встановлення наявності залишкових дефектів 2-го рівня критичності (помірних дефектів).

Якщо відношення сумарного значення дефектів 2-го рівня критичності (помірних дефектів) до загальної кількості виявлених під час базового тестування дефектів дорівнює або перевищує 0.5, то приймається висновок про встановлення наявності залишкових дефектів 3-го рівня критичності (серйозних дефектів). Якщо кількість дефектів 3-го рівня критичності (серйозних) перевищує 2, то приймається висновок про встановлення наявності залишкових дефектів 4-го рівня критичності (катастрофічних дефектів). Якщо кількість дефектів 4-го рівня критичності (катастрофічних) перевищує або дорівнює 1, то приймається висновок про непридатність ПЗ і можливу відмову системи. Отже, встановлено, що пороги мають наступні значення: 0.75; 0.5; 2; 1.

3 МЕТОД ІДЕНТИФІКАЦІЇ ЗАЛИШКОВИХ ДЕФЕКТІВ У ПРОГРАМНОМУ ЗАБЕЗПЕЧЕННІ

3.1 Алгоритм ідентифікації залишкових дефектів у програмному забезпеченні

Метод ідентифікації залишкових дефектів неможливо створити без розробки алгоритму. При цьому враховано, що порогові коефіцієнти масиву *R* не залежать від загальної кількості виявлених під час базового тестування дефектів. На початку роботи (блок 1) відбувається зчитування інформації, отриманої при базовому тестуванні. Цю інформацію зберігають в окремому файлі (у репозиторії, хмарному сховищі, тощо) і вона являє собою двомірний масив даних. Далі зчитують дані з бази знань (блоки 2 та 3). Спочатку зчитують масив текстових формулювань висновків – *F* (блок 2), а потім масив порогів допустимої кількості дефектів – *R* (блок 3). Наступна дія – це створення на основі даних про тестування масиву *D* (блок 4). Цей масив містить сумарні значення дефектів окремо по кожному із рівнів критичності. Масив є одномірним і має чотири значення. Після цього, розраховують значення масиву – *DN* (блок 4), який містить значення для порівняння з порогами допустимої кількості дефектів з масиву – *R*. Далі послідовно перевіряють правила формування висновку (відповідно до підрозділу 2.3) (блоки 6-17). Спочатку порівнюють між собою перші значення масивів *DN* і *R* (умова порівняння – $dn_1 < r_1$) (блок 6). Якщо відповідь негативна, то першому значенню масиву результатів перевірки правил *K* присвоюється значення k_1=1 (блок 7), якщо позитивна – то k_1=0 (блок 8). Тобто, формується перше значення одномірного масиву результатів перевірки правил *K*. Далі так само перевіряють наступні правила (блоки 9-17), виконання яких призводить до формування наступних значень масиву *K*. Далі перетворюють поєднані значення масиву *K* з двійкового числа на десяткове (блок 18). Отримане десяткове значення – g є індексом, який дозволяє здійснити вибір з масиву текстових формулювань висновків – *F*. Обраний висновок – f_g і є підсумком роботи програми (блок 2) (рисунок 3.1).

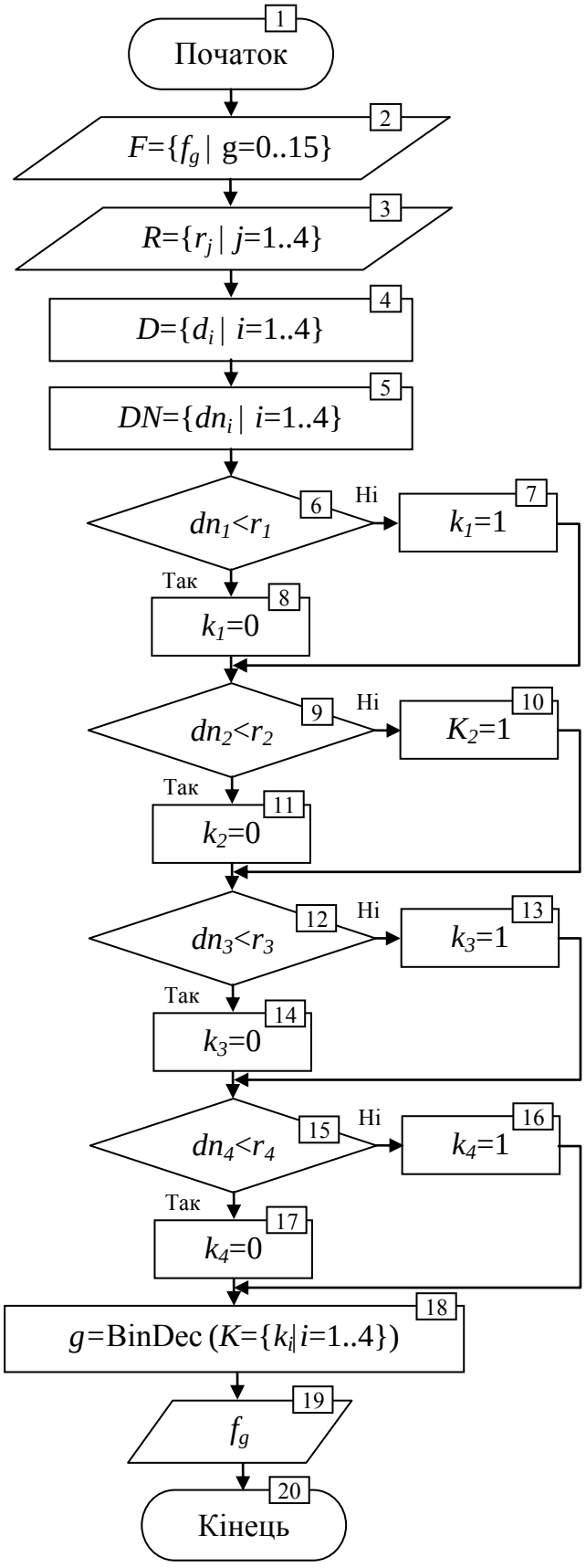


Рисунок 3.1 – Блок-схема методу ідентифікації залишкових дефектів у ПЗ

3.2 Модель і метод ідентифікації залишкових дефектів у програмному забезпеченні

Оскільки задачі прогнозування залишкових дефектів у ПЗ не притаманні властивості числового ряду, відсутні обернені зв'язки та немає потреби в класифікації та кластеризації даних, то для розв’язання такої задачі оберемо багатошаровий персептрон.

Структура цієї ШНМ представлена на рисунку 3.2.

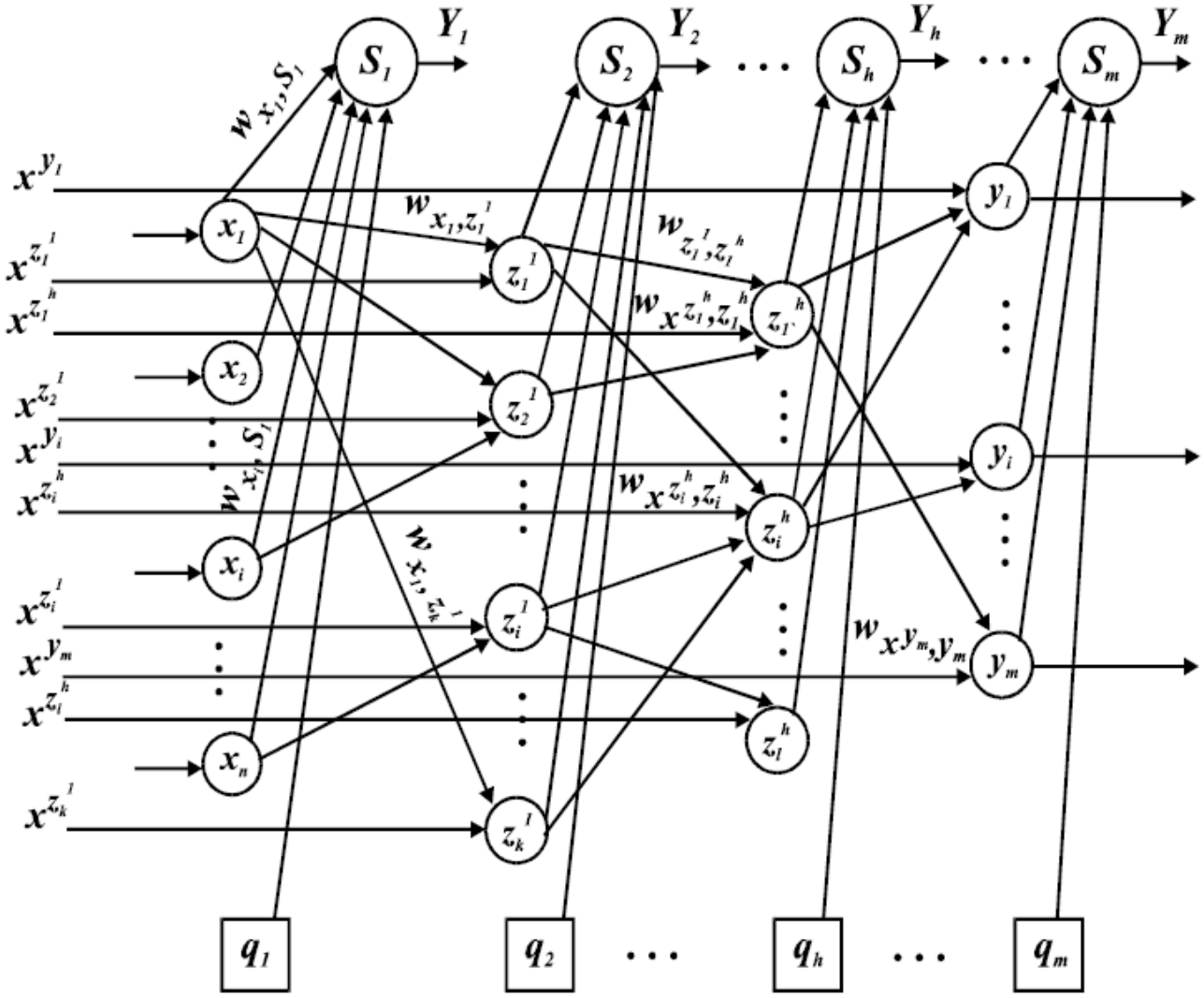


Рисунок 3.2 – Структура ШНМ для прогнозування залишкових дефектів у ПЗ та визначення їх рівня критичності

Нейрони вхідного (рецепторного) шару визначає природомовна інформація про кількість і типи дефектів, виявлених під час основного тестування, тобто звіт основного тестування – множина $X=\{x_i\}, i=(\overline{1,\ z_k})$, де x_i – i-й нейрон вхідного (рецепторного) шару (інформація з i-го рядка звіту основного тестування), кількість вхідних нейронів z_k – кількість рядків аналізованого звіту основного тестування.

Нейрони вихідного (ефекторного) шару визначають множину характеристик якості $Y=\{y_l\}, l=(\overline{1,\ 4})$, де y_l – l-й функціонал ефекторного шару багатошарового персептрону (l-ий рівень критичності), кількість вихідних нейронів наразі становить 4 (оскільки наразі розглядаються 4 рівні критичності залишкових дефектів).

Нейрони першого прихованого (апроксимуючого) шару прямонапрямленого персептрона складають множину $\overline{Z^1}=\{z_j^1\}, j=(\overline{1,k})$, де z_j^1 – j-й нейрон першого прихованого (апроксимуючого) шару, k – кількість нейронів першого прихованого (апроксимуючого) шару (достатньою є кількість нейронів першого прихованого шару $k=(z_k+l)/2$).

Нейрони другого прихованого (коригуючого) шару прямонапрямленого персептрона визначено множиною $\overline{Z^2}=\{z_g^2\}, g=(\overline{1,k_2})$, де z_g^2 – g-й нейрон другого прихованого (коригуючого) шару, k_2 – кількість нейронів другого прихованого (коригуючого) шару (достатньою є кількість нейронів першого прихованого шару $k_2=(z_k+l)/3$).

Вектор порогових величин зміщень множини нейронних елементів визначено як: $\overline{Q}=\{q_l\}, l=(\overline{1,\ 4})$, де q_l – зміщення l-го нейронного елемента, $l=4$ – кількість зміщень нейронних елементів (кількість рівнів критичності).

Ваги зв'язків відображено ваговими матрицями: $\overline{W_{X,Z^1}}=\left\langle w_{x_i,z_j^1}\right\rangle, i=(\overline{1,\ z_k}), j=(\overline{1,\ k})$, де w_{x_i,z_j^1} – ваговий коефіцієнт зв'язку

між x_i-м входом та z_j-м нейроном першого прихованого шару; $\overline{W_{Z^1,Z^2}} = \left\langle w_{z_j^1,z_g^2} \right\rangle$, $j = (\overline{1,k}), g = (\overline{1,k_2})$, де $w_{z_j^1,z_g^2}$ – ваговий коефіцієнт зв'язку між z_j-м нейроном апроксимуючого (першого) прихованого шару та z_g-м нейроном коригуючого (другого) прихованого шару; $\overline{W_{Z^2,Y}} = \left\langle w_{z_g^2,y_l} \right\rangle$, $l = (\overline{1,\ 4})$, де $w_{z_g^2,y_l}$ – ваговий коефіцієнт зв'язку між z_g-м нейроном коригуючого (другого) прихованого шару та l-м нейроном вихідного шару.

Формула для визначення l-го функціоналу ефекторного шару ШНМ y_l має вигляд:

$$y_l = f_1\left(f_2(Z^1)\cdot\left(\sum_{j=1}^{k}\sum_{g=1}^{k_2}(z_j^1 \cdot w_{z_j^1 z_g^2})\right) + \sum_{i=1}^{Z_k}(x_i \cdot w_{x_i,z_j^1})\right) - w_{z_g^2,y_1}, \qquad (3.1)$$

де f_1 – активаційна функція нейронів ефекторного шару ШНМ (лінійна функція, результати якої лежать в інтервалі [0; 1]), f_2 – активаційна функція нейронів прихованих шарів (гіперболічний тангенс).

Для опису текстового формування висновку про встановлення наявності залишкових дефектів введемо $f_g \in F$ (F={ f_g | g=0..15}), де f_g – текстове формулювання висновку про встановлення наявності або відсутності залишкових дефектів, де g – кількість текстових формулювань висновків, що змінюється від 0 до g.

Загальна кількість текстових формувань висновку не може перевищувати шістнадцять, оскільки кількість параметрів, що перевіряють дорівнює чотирьом (4^2=16). Проте, оскільки початкове g=0, кінцеве значення індексу g=15. Індекс g призначено для вибору з масиву текстових формулювань потрібного висновку.

Повний перелік текстових формулювань висновків про наявність залишкових дефектів у ПЗ та кодування правил формулювань висновків наведено у додатку Г, таблиці Г.1, Г.2.

Метод ідентифікації залишкових дефектів у програмному забезпеченні складається з наступних кроків:

1) на основі результатів роботи ШНМ відбувається формування множини $D = \{ d_i \mid i=1..4\}$, де d_i – сумарні значення дефектів кожного з рівнів критичності;

2) з використанням множини D відбувається формування множини $DN = \{ dn_i \mid i=1..4\}$. Перші два члени множини (dn_1, dn_2) є відношення $dn_i = dn_i / n$, де n – загальна кількість виявлених під час базового тестування дефектів ПЗ. Наступні два члени множини (dn_3, dn_4) є сумарні значення дефектів третього і четвертого рівнів критичності, відповідно $dn_3 = d_3$ а $dn_4 = d_4$;

3) з використанням множини R (R – множина порогів допустимої кількості дефектів), відбувається порівняння між собою значень елементів множин DN і R (умова порівняння – $dn_i < r_i$) та формування множини $K = \{ k_i \mid i=1..4\}$, призначеної для збереження результатів порівняння та формування чотирьохбітного двійкового числа;

4) отримання індексу g для текстової множини $F = \{ f_g \mid g=0..15\}$ шляхом перетворення чотирьохбітного двійкового числа в десяткове;

5) вибір з текстової множини F необхідного висновку про наявність або відсутність залишкових дефектів у ПЗ, виведення висновку користувачу та запис у файлі.

Отже, вдосконалено нейромережну модель ідентифікації залишкових дефектів у програмному забезпеченні на основі звіту основного тестування, яка відрізняється від відомих тим, що дає можливість враховувати важливість кожного рядка звіту основного тестування, а також взаємний вплив атрибутів в межах дефекту кожного рівня критичності. Вихідні функціонали ШНМ, що відповідають значенням рівнів критичності, дають можливість оцінити сумарний

вплив відшуканих під час основного тестування помилок на наявність залишкових дефектів у програмному забезпеченні.

Крім цього, розроблено метод ідентифікації залишкових дефектів у програмному забезпеченні, суть якого полягає у виявленні множини дефектів різних рівнів критичності та аналізу цієї множини на предмет наявності або відсутності залишкових дефектів. Метод відрізняється від відомих тим, що вхідна інформація про результати основного тестування опрацьовується штучною нейронною мережею.

3.3 Розроблення вимог до інформаційної технології ідентифікації залишкових дефектів у програмному забезпеченні

Як було зазначено у роботах [8-10, 49] однією з причин багатьох інцидентів та катастроф, пов'язаних із розробкою ПЗ, є недостатнє або некоректне написання вимог у специфікації щодо ПЗ. З [50-55] можна зробити висновок, що якість ПЗ залежить від специфікації вимог, а також від достатньої та повної інформації, яка зазначена у цих вимогах.

Відомо, що на сьогодні відповідність розробленого ПЗ вимогам специфікації розглядається на кінцевих етапах розробки. Це призводить до виявлення відсутності необхідної інформації у специфікації і до повторного її коригування. В зв'язку із такою ситуацією, можливі затримки у випуску готового ПЗ і до додаткових фінансових витрат. При відмові від повторного коригування специфікації, виникають інциденти та катастрофи у роботі ПЗ. Тому, перевірку показників для визначення метрик якості та складності ПЗ важливо виконувати на початкових стадіях ЖЦ ПЗ.

За дослідженнями [56, 57] значний відсоток дефектів ПЗ (10-23%) виникає на етапі формулювання вимог. Також на цьому етапі можливі інформаційні втрати завдяки формуванню неповних та нечітких вимог у специфікації. Особливо такі втрати притаманні проєктам, які розробляються у різних галузях і, відповідно, не можуть мати успішної реалізації. Тому, на етапі формування вимог

актуальним завданням є аналіз специфікації вимог до ПЗ, яке розробляється [56]. Для розробки специфікації, яка буде містити перелік усіх необхідних вимог, потрібно враховувати наступне:

– бізнес вимоги;

– стандарти інженерії ПЗ;

– стандарти предметної галузі;

– іншу документацію.

Далі, аналізуючи специфікацію вимог, необхідно зробити висновок про достатність чи недостатність інформації щодо якості у ПЗ, яке розробляється. При прийнятті рішення щодо достатності такої інформації, продовжують роботу над розробкою ПЗ. В іншому випадку – вирішують питання щодо доповнення цієї недостатньої інформації. При цьому, обов'язковим є врахування пріоритетності вхідної інформації. Для аналізу вимог специфікації зазвичай використовують метричний аналіз.

Згідно стандарту ISO/IEC 25010:2011, якість – це ступінь, до якого ПП або система можуть використовуватися окремими користувачами для задоволення своїх потреб для досягнення конкретних цілей з ефективністю, результативністю, свободою від ризику та задоволенням у конкретних умовах використання [50].

Властивості якості у використанні класифікуються за наступними характеристиками:

– ефективність (ефективність роботи, розробленого ПЗ);

– задоволення (корисність, довіра, задоволення, комфорт);

– свобода від ризику (зменшення економічного ризику, ризиків для здоров'я та безпеки, екологічних ризиків);

– охоплення контексту (повнота контексту, гнучкість) [50].

Згідно стандарту ISO/IEC 25010:2011 [50], модель якості продукту класифікує властивості якості продукту за вісьмома характеристиками, кожна із яких складається з набору взаємопов'язаних підхарактеристик:

– функціональна придатність (функціональна завершеність, правильність, відповідність);

– ефективність виконання (час поведінки, використання ресурсів, ємність);

– сумісність (співсумісність, сумісність);

– юзабіліті (доречність впізнаваність, можливість навчання, працездатність, захист від помилок користувача, естетика інтерфейсу користувача, доступність);

– надійність (зрілість, доступність, відмовостійкість, відновлюваність);

– безпека (конфіденційність, цілісність, відповідальність, автентичність);

– ремонтопридатність (модульність, багаторазове використання, аналізованість, модифікованість, перевіряємість);

– портативність (адаптивність, монтажність, замінність) [50].

Враховуючи перераховані характеристики, згідно стандарту ISO/IEC 25010:2011 [50], в роботі розроблено вимоги до інформаційної технології ідентифікації залишкових дефектів у ПЗ (таблиця 3.1).

Таблиця 3.1 – Вимоги до інформаційної технології ідентифікації залишкових дефектів у ПЗ

Вимога	Характеристика
1	2
Загальні	Система повинна забезпечувати ідентифікацію залишкових дефектів у ПЗ шляхом аналізу вихідного коду та/або використання тестових сценаріїв. Має бути забезпечена можливість автоматизованої ідентифікації та класифікації дефектів
Функціональності	Система може автоматично виявляти та визначати дефекти на всіх етапах розробки. Можливість інсталяції порогових значень для рівня важливості дефектів. Система повинна надавати інтерфейс для перегляду та аналізу ідентифікованих дефектів

Продовження таблиці 3.1

1	2
Продуктивності	Система працює швидко та ефективно, з обробкою великої кількості коду за короткий час. Максимальний час виявлення дефектів має бути встановлений (наприклад, не більше певної кількості годин після зміни в коді)
Надійності та безпеки	Захист від несанкціонованого доступу до інформації про ідентифіковані дефекти. Забезпечення стійкості системи до помилок та забезпечення її надійності
Інтеграції та невід'ємності	Система має інтегруватися з існуючими засобами розробки та тестування. Забезпечення сумісності з іншими платформами та середовищем розробки
Моніторингу та звітності	Можливість вести журнал ідентифікованих дефектів та їх статус. Генерація звітів для розробників, тестувальників та керівництва
Керування ЖЦ	Можливість оновлення та підтримки системи ідентифікації дефектів протягом усього ЖЦ ПЗ
Документації	Надання детальної документації з використання, налаштування та адміністрування систем
Тестування	Вимоги до тестування інформаційної технології ідентифікації. Визначення критеріїв успішності для тестування
Автоматизації	Система підтримує автоматизовану ідентифікацію та аналіз коду без значного втручання тестувальника. Можливість налаштування автоматичних сповіщень та попереджень про ідентифікацію нових дефектів

Кінець таблиці 3.1

1	2
Адаптованості	Система повинна бути адаптована до різних типів ПЗ, мовного програмування та архітектурних підходів. Забезпечення можливості розширення функціоналу для підтримки нових технологій та програмування версій мов
Зручності використання	Інтуїтивний інтерфейс для тестувальників різних рівнів кваліфікації. Підтримка можливостей фільтрації, сортування та пошуку в інтерфейсі користувача
Ресурсного споживання	Мінімізація впливу на продуктивність розробників та тестувальників під час роботи із системною ідентифікацією
Доступності	Забезпечення доступу до системи для користувачів із необхідними вимогами до доступності
Сумісності з існуючими стандартами	Відповідність існуючим стандартам безпеки щодо обробки та збереження конфіденційної інформації
Забезпечення конфіденційності	Заходи для захисту конфіденційної інформації, такої як результати ідентифікації дефектів

3.4 Висновки

У третьому розділі вдосконалено нейромережну модель ідентифікації залишкових дефектів у програмному забезпеченні на основі звіту основного тестування, яка відрізняється від відомих тим, що дає можливість враховувати важливість кожного рядка звіту основного тестування, а також взаємний вплив атрибутів в межах дефекту кожного рівня критичності. Вихідні функціонали ШНМ, що відповідають значенням рівнів критичності, дають можливість оцінити

сумарний вплив відшуканих під час основного тестування помилок на наявність залишкових дефектів у програмному забезпеченні.

Крім цього, розроблено метод ідентифікації залишкових дефектів у програмному забезпеченні, суть якого полягає у виявленні множини дефектів різних рівнів критичності та аналізу цієї множини на предмет наявності або відсутності залишкових дефектів. Метод відрізняється від відомих тим, що вхідна інформація про результати основного тестування опрацьовується штучною нейронною мережею.

В третьому розділі також розроблено вимоги до інформаційної технології ідентифікації залишкових дефектів у програмному забезпеченні.

4 ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ІДЕНТИФІКАЦІЇ ЗАЛИШКОВИХ ДЕФЕКТІВ У ПРОГРАМНОМУ ЗАБЕЗПЕЧЕННІ

4.1 Формалізація вимог до інформаційної технології ідентифікації залишкових дефектів у програмному забезпеченні

Формалізація вимог до інформаційної технології ідентифікації залишкових дефектів у програмному забезпеченні є важливою частиною процесу розробки ПЗ. Процес формалізації включає в себе декілька кроків [58-62]: аналіз вимог користувача; визначення критеріїв ідентифікації дефектів; розробка тестових сценаріїв; визначення метрик якості; використання інструментів автоматизації тестування; перевірка відповідності стандартам. Перераховані кроки є загальними, тому їх потрібно адаптувати до інформаційної технології ідентифікації залишкових дефектів у ПЗ. Аналіз вимог користувача було проведено у розділі 3. Відповідно до таблиці 3.1, наведено вимоги до інформаційної технології ідентифікації залишкових дефектів у ПЗ. Це дозволило створити масив вимог, який вміщує 16-ть вимог (таблиця 4.1).

Таблиця 4.1 – Масив вимог до інформаційної технології ідентифікації залишкових дефектів у ПЗ

Код вимоги	Вимога	Код вимоги	Вимога
0	Загальні	8	Тестування
1	Функціональності	9	Автоматизації
2	Продуктивності	10	Адаптованості
3	Надійності та безпеки	11	Зручності використання
4	Інтеграції та невід'ємності	12	Ресурсного споживання
5	Моніторингу та звітності	13	Доступності
6	Керування ЖЦ	14	Сумісності з стандартами
7	Документації	15	Забезпечення конфіденційності

Представлені у таблиці 4.1 вимоги стосуються функціонування запропонованого ПП протягом усього ЖЦ. Проте, на початку ЖЦ необхідно виявити роботоздатність запропонованого ПП. Основною функцією є ідентифікація залишкових дефектів у ПЗ. Проте, всебічно і максимально повно перевірити функціональність можливо лише з використанням діаграми зміни станів вихідних даних програми ідентифікації залишкових дефектів у ПЗ (рисунок 4.1).

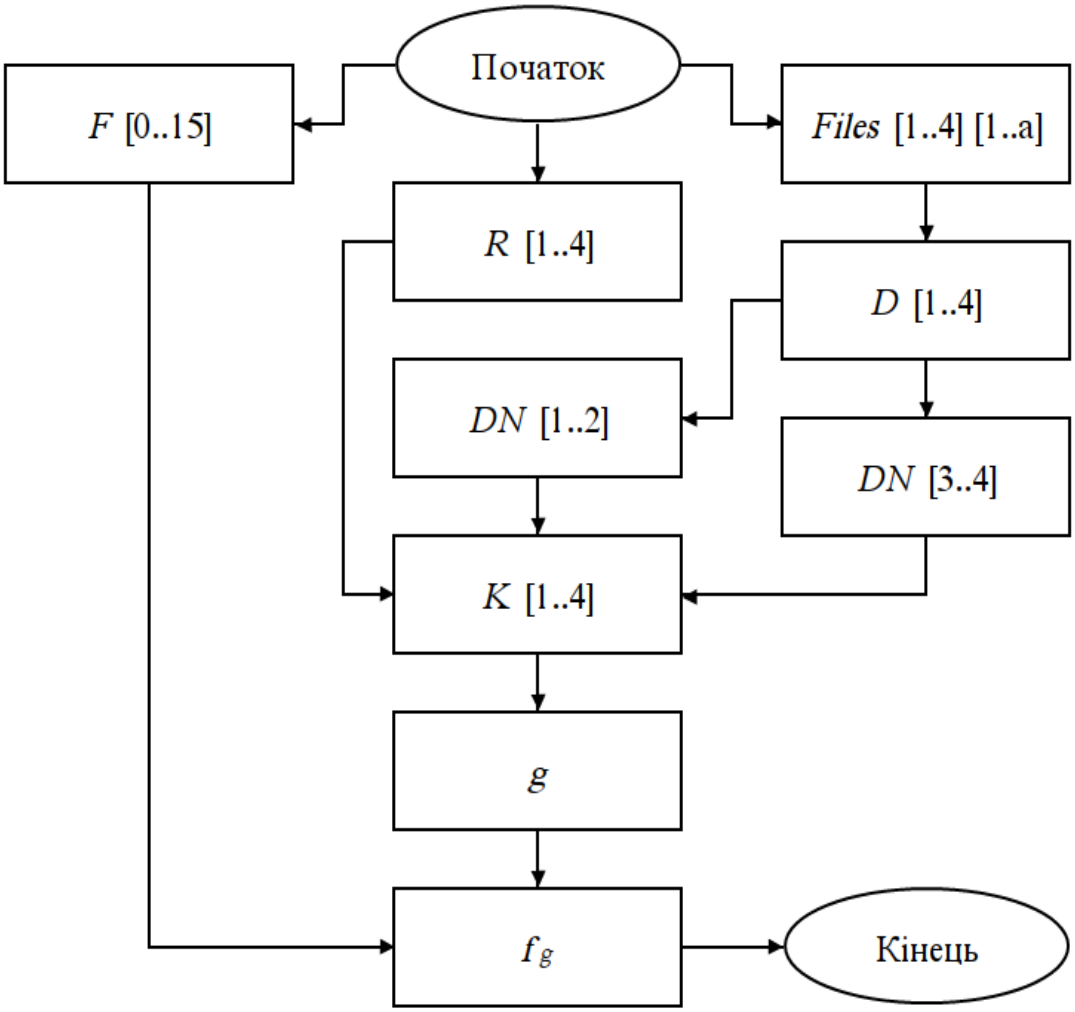


Рисунок 4.1 – Діаграма зміни станів вихідних даних програми ідентифікації залишкових дефектів у ПЗ

Діаграма станів показує, що після початку роботи програми відбувається перше перетворення і виділені комірки пам'яті заповнюються даними

(масиви *Files F* та *R*). Ці дані містять усю необхідну інформацію для правильного функціонування програми:

– вихідні дані тестування (двомірний масив *Files* [1..4][1..a]);

– порогові коефіцієнти (одномірний масив *R* [1..4]);

– текстові правила (одномірний масив *F* [0..15]);

Хоча діаграма станів показує, що зчитування даних проходить паралельно, проте в реальній програмі ця дія проходить послідовно. Це збільшує час на першу зміну, проте гарантує правильне заповнення комірок пам'яті (оскільки наступні дані розташовуються в пам'яті тільки після закінчення роботи з попередніми).

Другою зміною стану даних є створення на основі даних про тестування *Files* масиву *D*, який містить сумарні значення дефектів окремо по кожному із рівнів критичності. Масив *D* є одномірним і має чотири значення.

Наступною зміною є створення масиву *DN*, який містить значення для порівняння з порогами допустимої кількості дефектів. Причому, формування елементів масиву проводиться двома різними шляхами. Формування перших двох елементів масиву пов'язане з розрахунками *DN* [1..2]. Два наступних елементи утворюються шляхом присвоювання *DN* [3..4].

Далі, при зміні перевіряють чотири правила формування висновку про встановлення наявності залишкових дефектів та формують масив *K* [1..4]. Зміна стану робиться послідовно, починаючи з першого і закінчуючи четвертим значенням. В процесі приймає участь масив порогових коефіцієнтів *R*. Слід зазначити, що отриманий масив *K* містить бінарні значення (0 або 1).

Наступною зміною стану є поєднання значень масиву *K* та перетворення отриманого двійкового числа на десяткове. Отримане десяткове значення *g* це індексом.

Остання зміна стану даних виконується з використанням масиву текстових формулювань висновків *F*. Результатом зміни є обраний висновок f_g. Після цього відбувається завершення роботи програми.

Критерії ідентифікації дефектів для вимог розрізняються. Тому, встановлення чітких критеріїв потрібно зробити для розробки ефективних методів ідентифікації, за якими можна визначити наявність дефекту.

Основою подальшої формалізації вимог служать тестові документи [63-67], використання яких дозволяє значно скоротити час на формалізацію вимог до інформаційної технології ідентифікації залишкових дефектів у програмному забезпеченні і робить цей процес стандартним.

4.2 Структура інформаційної технології ідентифікації залишкових дефектів у програмному забезпеченні

Інтелектуальна інформаційна технологія ідентифікації залишкових дефектів у ПП базується на таких принципах проєктування та функціонування [68-73]:

– принцип розвитку (оновлення складу і функцій інформаційної технології без порушення її функціонування);

– принцип сумісності (можливість взаємодії інформаційної технології з іншими інформаційними технологіями);

– принцип автоматизації опрацювання інформації (використання технічних інструментів на всіх стадіях проходження інформації);

– принцип ефективності (максимальний ефект при мінімальних витратах);

– принцип етапності (можливість послідовного розвитку інформаційної технології);

– принцип системності (єдиний методологічний підхід, який розглядає об'єкт дослідження як єдине ціле);

– принцип відкритості (забезпечення правдивості, достовірності, оперативності, регулярності та надійності інформації).

З урахуванням перерахованих принципів проєктування та функціонування в роботі розроблено структуру формування інформаційної технології ідентифікації залишкових дефектів у ПП, рисунок 4.2.

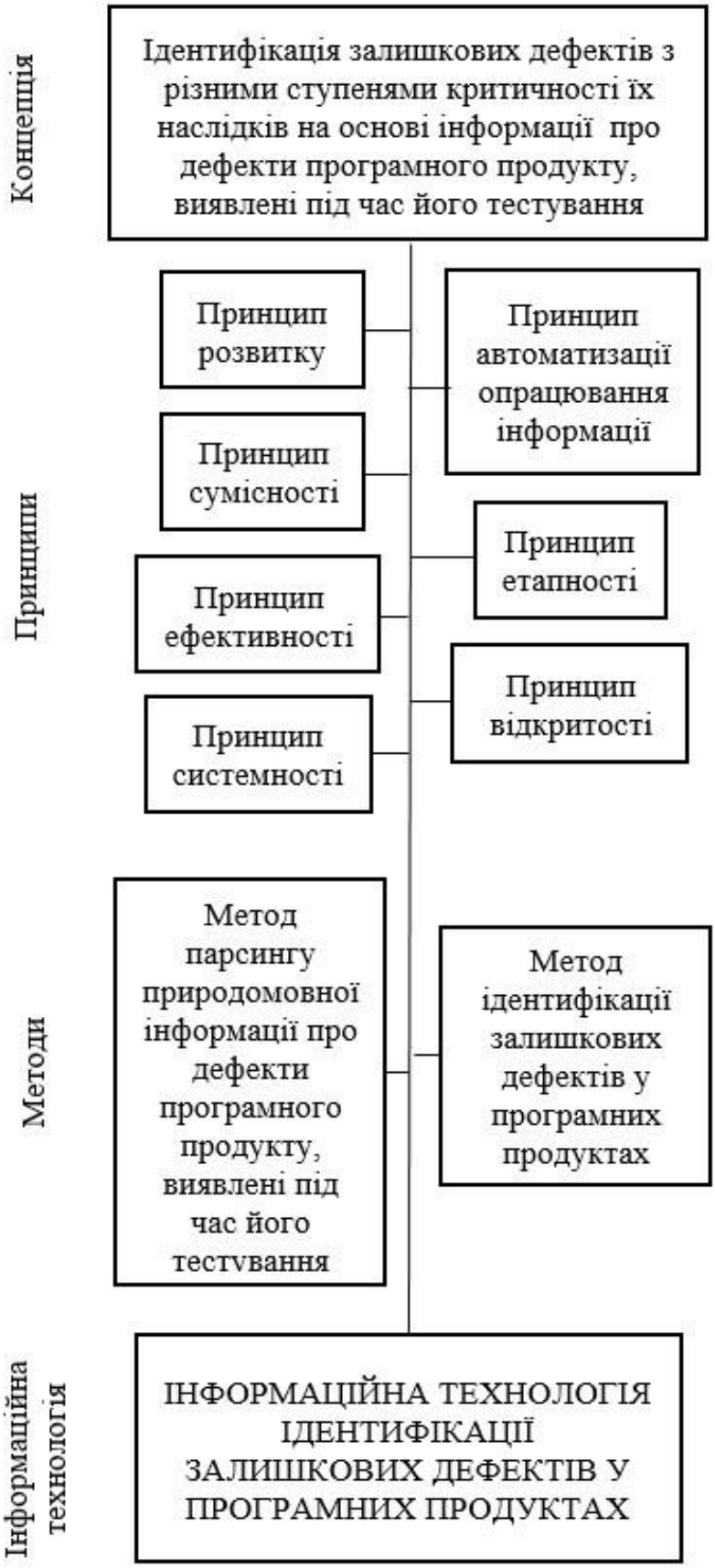


Рисунок 4.2 – Структура формування інформаційної технології ідентифікації залишкових дефектів у ПП

Використаємо метод парсингу природомовної інформації про дефекти ПП, виявлені під час його тестування, та метод ідентифікації залишкових дефектів у програмних продуктах [39]. Деталізована схема інформаційної технології ідентифікації залишкових дефектів у ПП представлена на рисунку 4.3.

Рисунок 4.3 – Деталізована схема інтелектуальної інформаційної технології ідентифікації залишкових дефектів у ПП

Інтелектуальна інформаційна технологія ідентифікації залишкових дефектів у ПЗ забезпечує висновок про відсутність залишкових дефектів або про наявність залишкових дефектів з різними ступенями критичності їх наслідків на основі природомовного звіту про дефекти ПЗ, виявлені під час його базового тестування.

Як інструмент інформаційної технології ідентифікації залишкових дефектів у ПП запропонуємо систему розробки бездефектного ПЗ шляхом встановлення наявності залишкових дефектів, структурна схема якої представлена на рисунку 4.4.

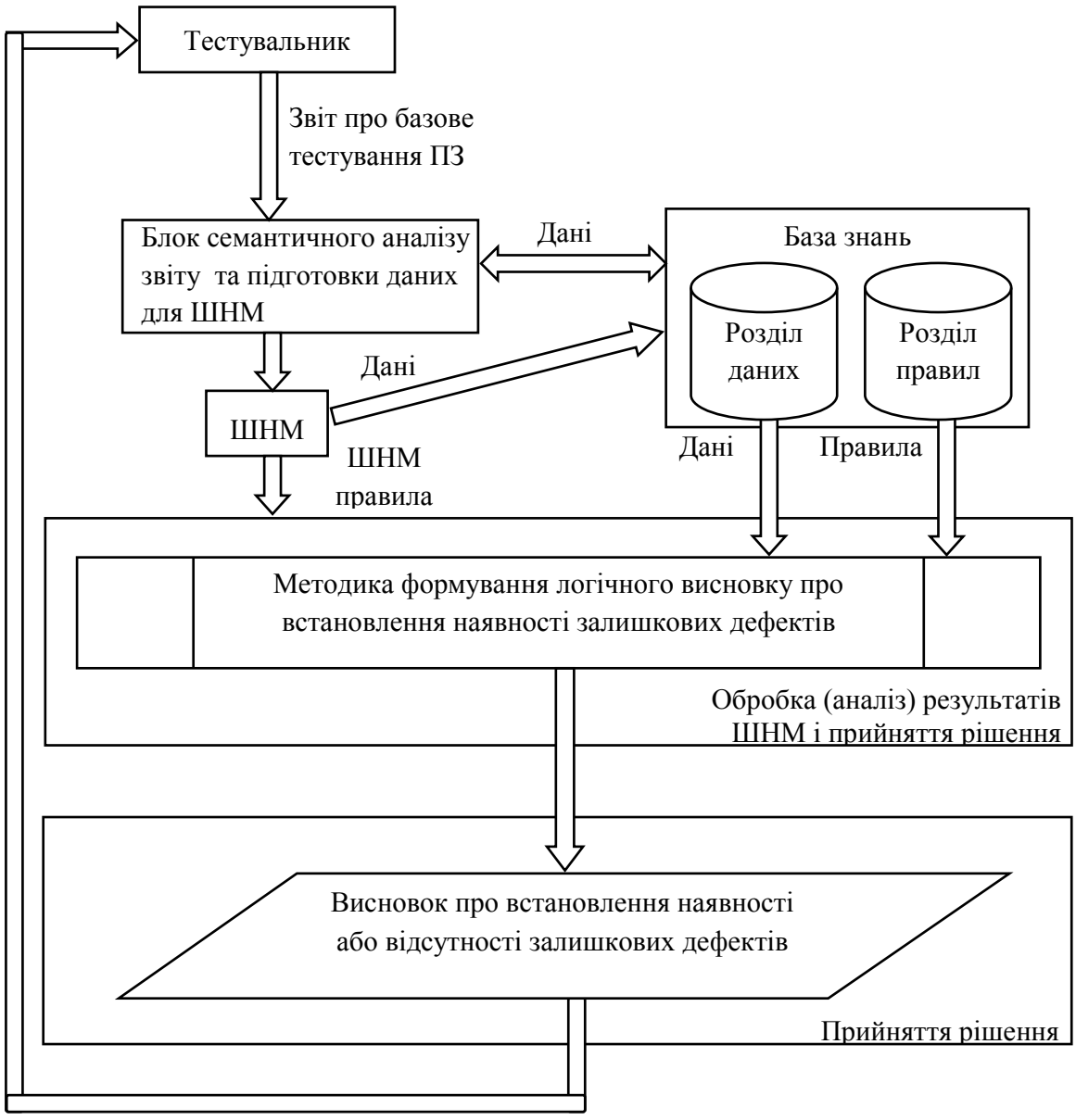


Рисунок 4.4 – Структура системи розробки бездефектного ПЗ шляхом встановлення наявності залишкових дефектів

Система розробки бездефектного ПЗ шляхом встановлення наявності залишкових дефектів функціонує наступним чином. Користувач системи на вхід подає звіт основного тестування.

Блок семантичного парсингу звіту та підготовки даних для ШНМ виконує семантичний парсинг звіту, вибираючи з нього інформацію про знайдені під час основного тестування дефекти ПЗ, після чого виконує перетворення цієї інформації з лінгвістичної форми представлення в кількісну форму за допомогою таблиць даних бази знань.

База знань містить таблиці присвоєння номерів типам виявлених дефектів ПЗ. База знань містить також вище розроблені правила формування висновку про встановлення наявності залишкових дефектів. База знань складається з:

– розділу даних, який буде містити інформацію про присвоєння номерів методам основного тестування (при виявленні помилок одного рівня категорійності, методи будуть об’єднуватись з присвоєнням одного номера); присвоєння номерів операціям тестування ПП, типам дефектів, рівням критичності прихованих дефектів; кількісне представлення вхідних даних результуючих векторів ШНМ; відповідність методу тестування, операцій тестування і типів виявлених під час тестування дефектів; відповідність між номером методів тестування ПП та рівнем критичності неідентифікованих дефектів; відповідність між операціями тестування ПЗ та рівнем критичності прихованих помилок;

– розділ правил для формування висновку відсутність чи наявність залишкових дефектів у програмному забезпеченні.

ШНМ опрацьовує всю отриману в кількісній формі інформацію про дефекти ПЗ, виявлені під час основного тестування, після чого надає інформацію про рівні критичності виявлених дефектів (0 або 1 по кожному з 4-х рівнів, де 1 означає приналежність дефекту саме цьому рівню критичності). Результати ШНМ записуються в таблицю даних бази знань.

Далі, згідно із вище розробленим методом ідентифікації залишкових дефектів у програмному забезпеченні відбувається формування висновку про

відсутність чи наявність залишкових дефектів у програмному забезпеченні (із зазначенням рівня(ів) критичності залишкових дефектів у разі встановлення їх наявності), який видається користувачу системи. Отже, запропонована система розробки бездефектного ПЗ шляхом встановлення наявності залишкових дефектів дозволяє користувачу, на основі звіту про результати основного тестування, одержати висновок про відсутність чи наявність залишкових дефектів у програмному забезпеченні (із зазначенням рівня(ів) критичності залишкових дефектів у разі встановлення їх наявності).

4.3 Результати функціонування інформаційної технології ідентифікації залишкових дефектів у програмному забезпеченні

Було проведено декілька експериментів із запропонованою системою розробки бездефектного ПЗ шляхом встановлення наявності залишкових дефектів. Під час проведення першого експерименту розробленою інформаційною технологією опрацьовано природомовну інформацію про дефекти довідково-інформаційної програмної системи «Ресторан», виявлені під час її базового тестування. Було виконано парсинг цієї природомовної інформації, вибрано інформацію про виявлені дефекти ПП під час його базового тестування, а також перетворено цю інформацію з лінгвістичної форми представлення в кількісну форму з використанням таблиць даних бази знань та сформовано вхідні вектори ШНМ. ШНМ опрацювала отриману вхідну інформацію та надала наступні результати у вигляді двомірного масиву:

Data1=[[0;0;1;0],[1;0;0;0],[1;0;0;0],[0;1;0;0],
[0;1;0;0],[0;1;0;0],[1;0;0;0],[1;0;0;0]]

Як видно з даних, розмір масиву 4 × 8. Тобто, чотири стовпця і вісім рядків. Кількість рядків масиву залежить від кількості тестувань. В розробленій програмі кількість рядків обмежена. Максимальна кількість рядків масиву в програмі дорівнює сто. Кількість стовпців є незмінною, оскільки кожен стовпець відповідає рівню критичності дефектів (яких є чотири: 1 – незначні; 2 – помірні, 3 – серйозні,

4 – катастрофічні). Дані масиву Data1 були збережені в текстовому файлі з назвою – «Ресторан.txt», який було розташовано у хмарному сховищі із загальним але обмеженим для користувачів доступом (Google Shared Driver [74]). Вміст файлу «Ресторан.txt» відображено на рисунку 4.5.

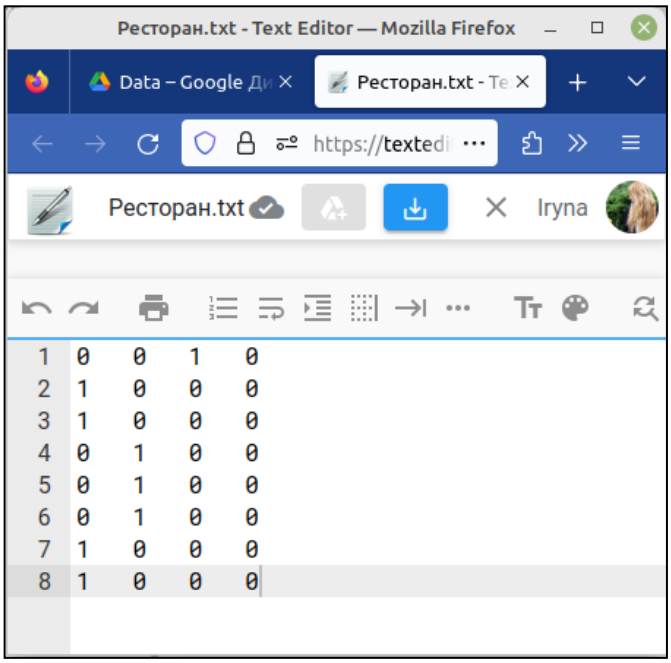


Рисунок 4.5 – Вміст файлу «Ресторан.txt», розташованому у хмарному сховищі Google Shared Driver

Розроблену в кваліфікаційній роботі програму ідентифікації залишкових дефектів у ПЗ користувач може активувати віддалено або локально. Три версії програми розташовано у хмарному сховищі Google Shared Driver. Для локальної активації потрібно завантажити необхідну версію програми.

Для операційних систем з встановленим Microsoft Office Excel файл з розширенням – .xlsx, для операційних систем з встановленим Libre Office Calc файл з розширенням – .ods, а для операційних систем, в яких не встановлено Office можна здійснити віддалену активацію програми ідентифікації залишкових дефектів у ПЗ. Подальший опис роботи розробленого автором ПЗ буде проведено з віддаленим способом активації програми. Для активації потрібно запустити файл з назвою «Аналіз» без розширення (рисунок 4.6).

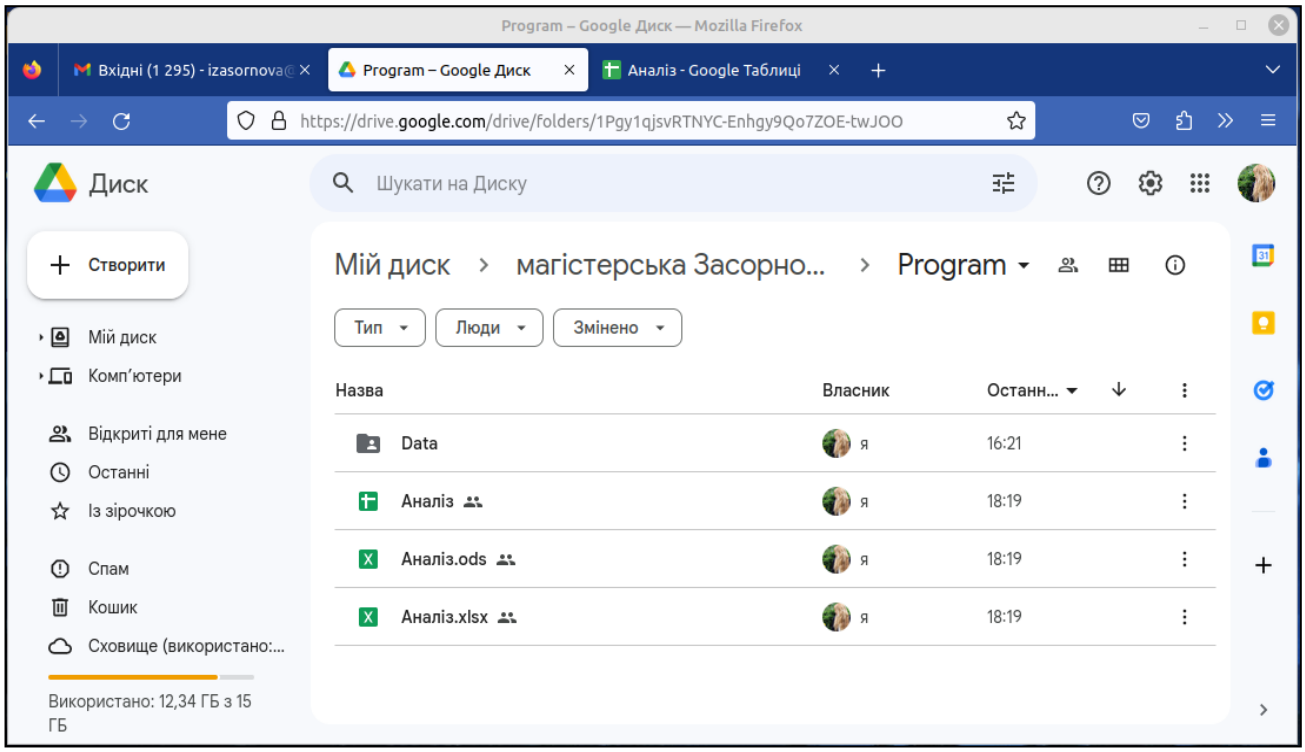


Рисунок 4.6 – Вміст теки «Program» хмарного сховища Google Shared Driver

Після активації відкриється перша сторінка – «Аналіз» програми ідентифікації залишкових дефектів у ПЗ (рисунок 4.7).

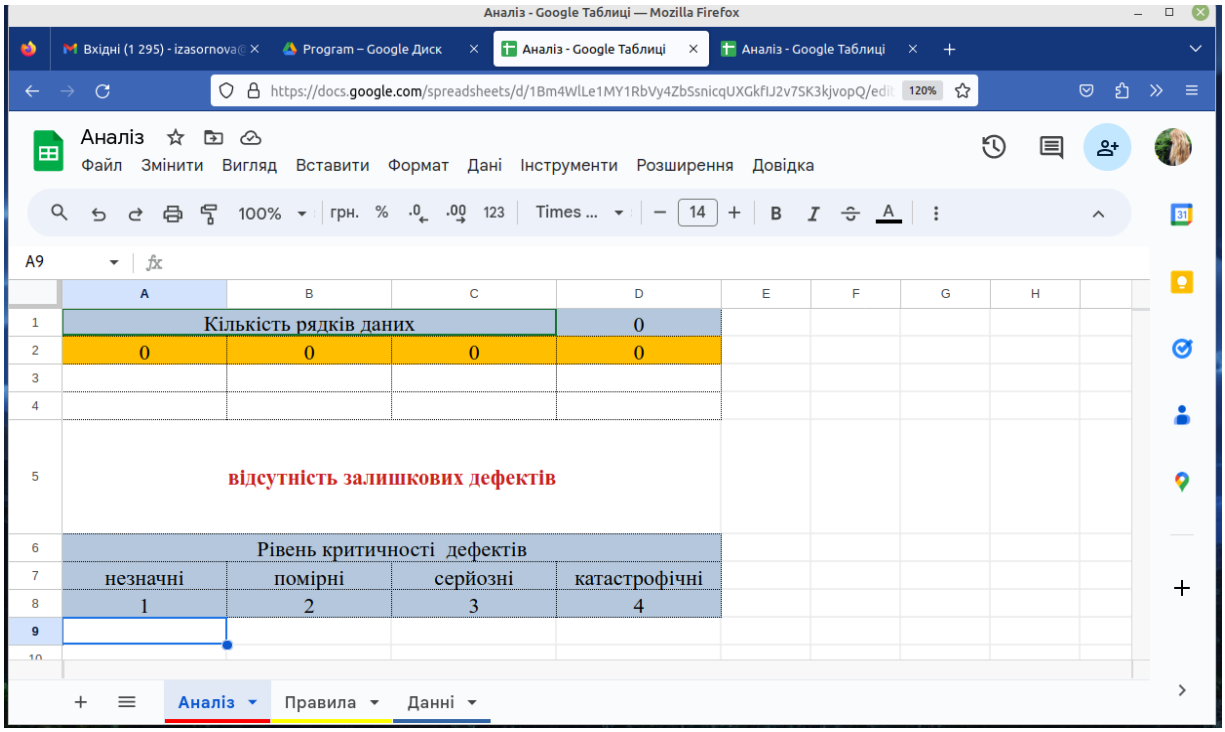


Рисунок 4.7 – Зовнішній вигляд першої сторінки програми

Далі потрібно імпортувати дані із зовнішнього файлу з розширенням txt. Для цього виконуємо вибір необхідної опції з верхнього меню програми «Файл», далі натискаємо – «Імпортувати». У вікні, що відкрилося обираємо потрібний файл та натискаємо кнопку «Додати» (рисунок 4.8).

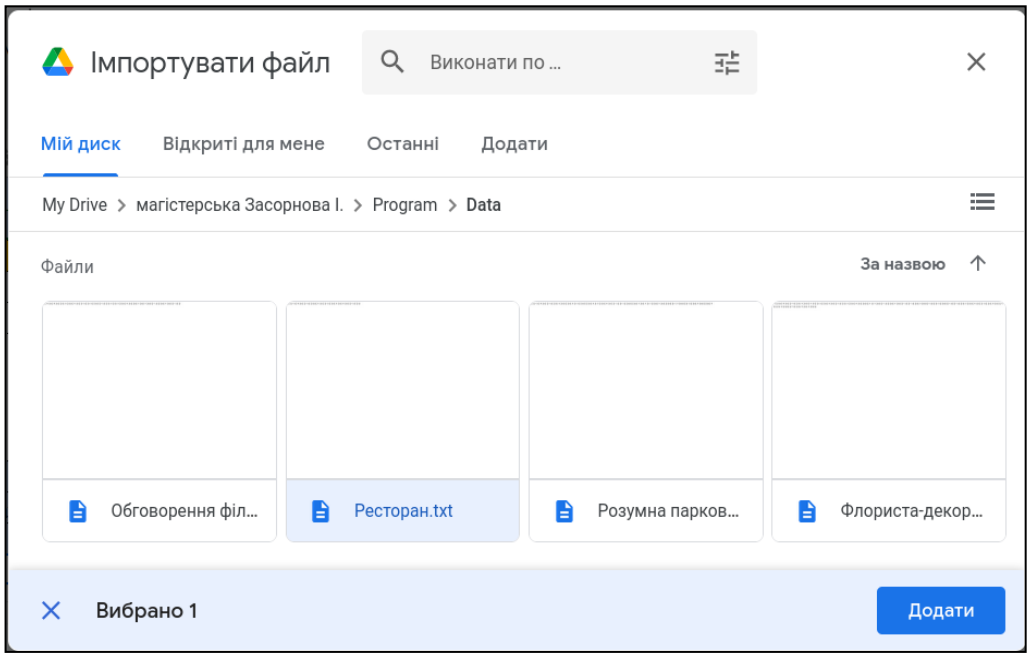


Рисунок 4.8 – Зовнішній вигляд вікна «Файл»

У наступному вікні, що відкрилося, обираємо опцію «Замінити дані» з вибраної клітинки, обираємо потрібний файл та натискаємо кнопку «Імпортувати дані» (рисунок 4.9).

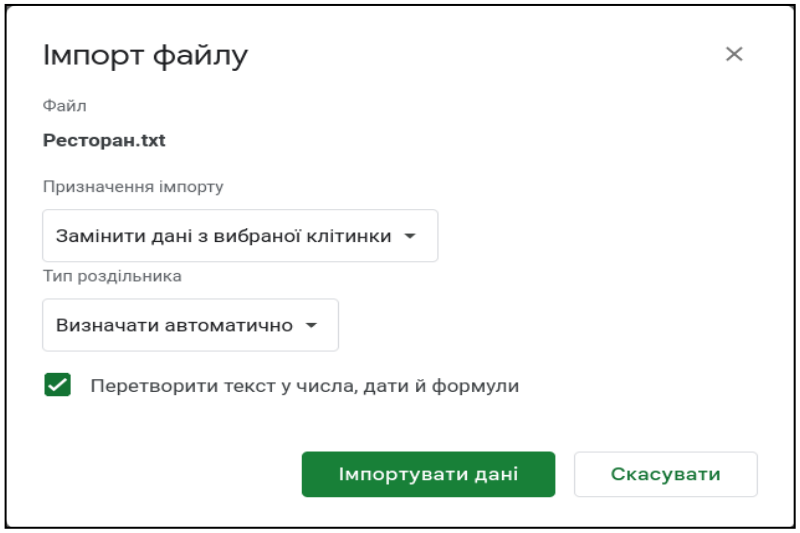
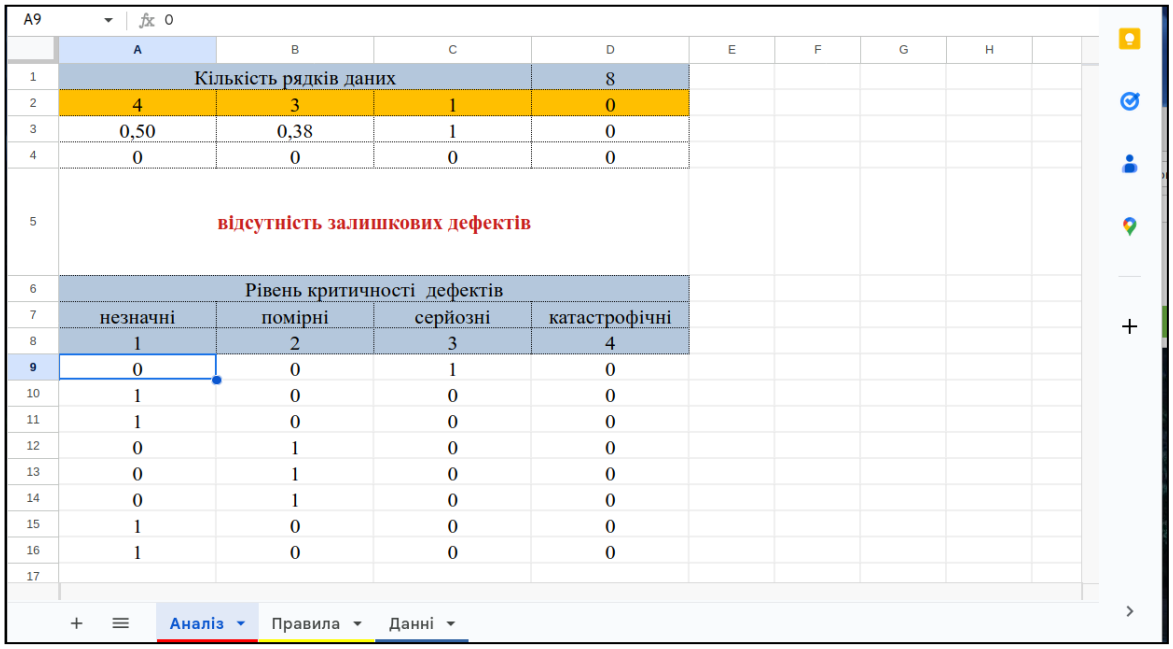


Рисунок 4.9 – Зовнішній вигляд вікна «Імпорт файлу»

Після цього, дані будуть імпортовані у програму.

Далі програма автоматично обробить дані про дефекти довідково-інформаційної програмної системи «Ресторан», виявлені під час її базового тестування та зробить висновок про відсутність залишкових дефектів (рисунок 4.10).



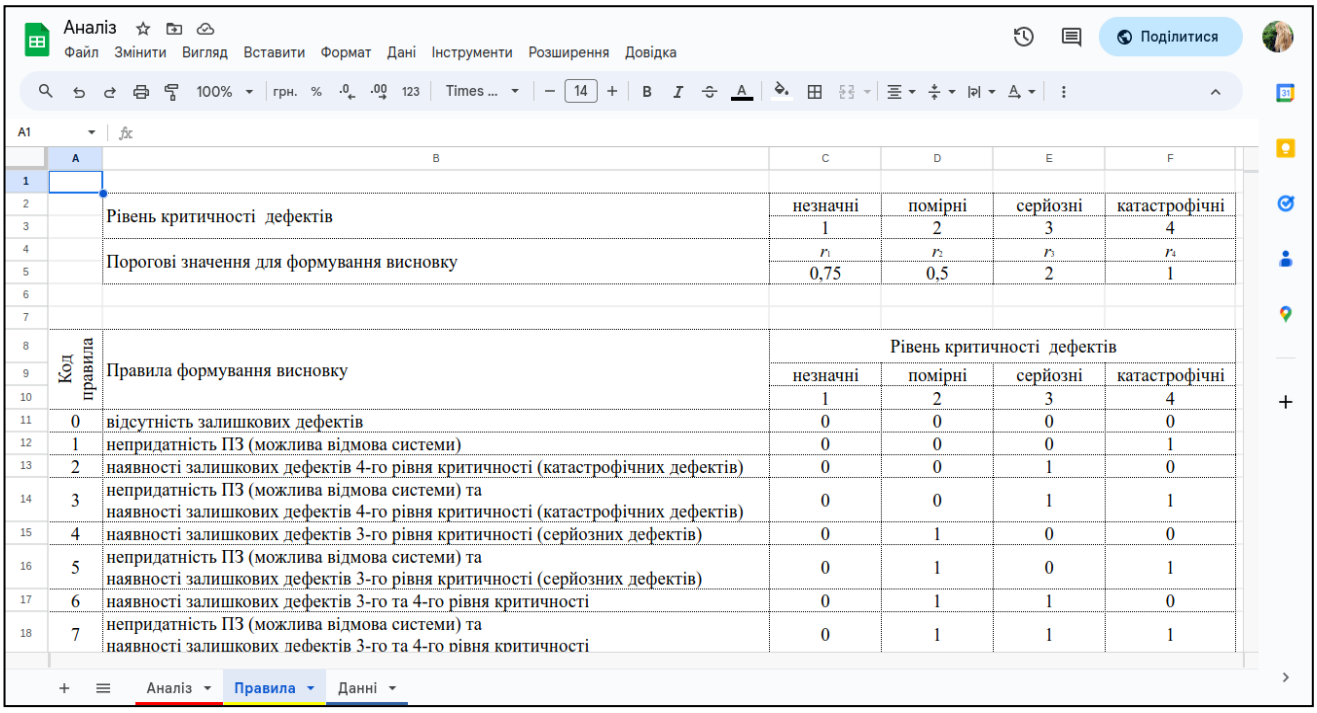
	A	B	C	D
1	Кількість рядків даних			8
2	4	3	1	0
3	0,50	0,38	1	0
4	0	0	0	0
5	відсутність залишкових дефектів			
6	Рівень критичності дефектів			
7	незначні	помірні	серйозні	катастрофічні
8	1	2	3	4
9	0	0	1	0
10	1	0	0	0
11	1	0	0	0
12	0	1	0	0
13	0	1	0	0
14	0	1	0	0
15	1	0	0	0
16	1	0	0	0
17				

Рисунок 4.10 – Зовнішній вигляд програми з представленням висновків про наявність залишкових дефектів у системі «Ресторан»

З рисунку 4.10 видно, що у системі «Ресторан» діагностовано чотири дефекти 1-го рівня критичності (незначні), три дефекти 2-го рівня критичності (помірні) та один дефект 3-го рівня критичності (серйозні).

Згідно розробленого методу, програма шляхом встановлення наявності залишкових дефектів, сформувала масив D = (4, 3, 1, 0) та DN = (0,5, 0,38, 1, 0).

Програма проаналізувала усі правила формування висновку про встановлення наявності залишкових дефектів, за введеними в базу даних значеннями, сторінка «Правила» (рисунок 4.11).



	Рівень критичності дефектів	незначні	помірні	серйозні	катастрофічні
		1	2	3	4
	Порогові значення для формування висновку	r_1	r_2	r_3	r_4
		0,75	0,5	2	1

Код правила	Правила формування висновку	Рівень критичності дефектів			
		незначні	помірні	серйозні	катастрофічні
		1	2	3	4
0	відсутність залишкових дефектів	0	0	0	0
1	непридатність ПЗ (можлива відмова системи)	0	0	0	1
2	наявності залишкових дефектів 4-го рівня критичності (катастрофічних дефектів)	0	0	1	0
3	непридатність ПЗ (можлива відмова системи) та наявності залишкових дефектів 4-го рівня критичності (катастрофічних дефектів)	0	0	1	1
4	наявності залишкових дефектів 3-го рівня критичності (серйозних дефектів)	0	1	0	0
5	непридатність ПЗ (можлива відмова системи) та наявності залишкових дефектів 3-го рівня критичності (серйозних дефектів)	0	1	0	1
6	наявності залишкових дефектів 3-го та 4-го рівня критичності	0	1	1	0
7	непридатність ПЗ (можлива відмова системи) та наявності залишкових дефектів 3-го та 4-го рівня критичності	0	1	1	1

Рисунок 4.11 – Зовнішній вигляд сторінки «Правила»

Тобто, було встановлено наступне:

правило 1 не спрацьовує $dn_1 < r_1$;

правило 2 не спрацьовує $dn_2 < r_2$;

правило 3 не спрацьовує $dn_3 < r_3$;

правило 4 не спрацьовує $dn_4 < r_4$.

Отже, жодне правило не спрацювало, тому програма видала висновок: «відсутність залишкових дефектів» (рисунок 4.10).

Тобто, можна зробити висновок, що у ПП «Ресторан» (1-ший експеримент) відсутні залишкові дефекти.

Слід зазначити, що окрім порогових значень для формування висновку, БД (сторінка «Правила», рисунок 4.11) містить текстові дані множини F, з якої програма здійснює вибір текстового висновку.

Під час другого експерименту, тестувальник подав звіт про дефекти програмної частини кіберфізичної частини системи «Розумна парковка», виявлені під час її базового тестування.

Блок семантичного аналізу звіту та підготовки даних для ШНМ виконав семантичний парсинг звіту, вибираючи з нього інформацію про знайдені під час

базового тестування дефекти ПП, після чого виконав перетворення цієї інформації з лінгвістичної форми представлення в кількісну форму за допомогою таблиць даних бази знань.

ШНМ опрацювала інформацію про дефекти ПЗ, виявлені під час базового тестування, після чого надала наступні результати (розмір масиву 4 × 20):

Data2=[[0;0;1;0],[0;1;0;0],[1;0;0;0],[0;1;0;0],[1;0;0;0],
[1;0;0;0],[1;0;0;0],[0;1;0;0],[0;1;0;0],[0;1;0;0],
[1;0;0;0],[0;0;1;0],[0;1;0;0],[1;0;0;0],[1;0;0;0],
[0;1;0;0] [1;0;0;0] [0;1;0;0] [0;1;0;0] [0;1;0;0]]

Отже, враховуючи надані ШНМ результати, очевидно, що ідентифіковано два дефекти з 3-м ступенем критичності їх наслідків (серйозних), десять дефектів з 2-м ступенем критичності їх наслідків (помірних) та вісім дефектів з 1-м ступенем критичності їх наслідків (незначних).

Оскільки, відношення сумарного значення дефектів з 1-м ступенем критичності їх наслідків до загальної кількості виявлених дефектів ПП дорівнює 8/20=0.4, тобто не перевищує 0.75, то залишкові дефекти з 2-м ступенем критичності їх наслідків в аналізованому ПП відсутні.

Оскільки, відношення сумарного значення дефектів з 2-м ступенем критичності їх наслідків до загальної кількості виявлених дефектів ПП дорівнює 10/20=0.5, то в аналізованому ПП наявні залишкові дефекти з 3-м ступенем критичності їх наслідків.

Оскільки, сумарне значення дефектів з 3-м ступенем критичності їх наслідків дорівнює 2, то в аналізованому ПП наявні залишкові дефекти з 4-м ступенем критичності їх наслідків.

Отже, запропонована інтелектуальна інформаційна технологія ідентифікації залишкових дефектів у ПП надає висновок про наявність залишкових дефектів з 3-м та 4-м ступенями критичності їх наслідків у програмній частині кіберфізичної системи «Розумна парковка» (рисунок 4.12).

Кількість рядків даних			20
8	10	2	0
0,40	0,50	2	0
0	1	1	0

наявності залишкових дефектів 3-го та 4-го рівня критичності

Рівень критичності дефектів			
незначні	помірні	серйозні	катастрофічні
1	2	3	4
0	0	1	0

Рисунок 4.12 – Зовнішній вигляд програми з представленням висновків про наявність залишкових дефектів у системі «Розумна парковка»

В якості третього експерименту розробленою інформаційною технологією опрацьовувалась природомовна інформація про дефекти програмної частини підсистеми керування освітленням кіберфізичної системи «Розумний будинок», виявлені під час її базового тестування.

Було виконано парсинг цієї природомовної інформації, вибрано інформацію про виявлені дефекти ПП під час його базового тестування, а також перетворено цю інформацію з лінгвістичної форми представлення в кількісну форму з використанням таблиць даних бази знань та сформовано вхідні вектори ШНМ. ШНМ опрацювала отриману вхідну інформацію та надала наступні результати (розмір масиву 4 × 49):

Data3=[[1;0;0;0],[1;0;0;0],[1;0;0;0],[1;0;0;0],[1;0;0;0],[1;0;0;0],[1;0;0;0],
[1;0;0;0],[1;0;0;0],[0;1;0;0],[1;0;0;0],[1;0;0;0],[0;1;0;0],[1;0;0;0],
[1;0;0;0],[0;1;0;0],[1;0;0;0],[0;1;0;0],[0;1;0;0],[0;1;0;0],[1;0;0;0],
[0;1;0;0],[1;0;0;0],[1;0;0;0],[0;1;0;0],[1;0;0;0],[0;1;0;0],[0;1;0;0],
[0;1;0;0],[1;0;0;0],[1;0;0;0],[0;1;0;0],[1;0;0;0],[1;0;0;0],[1;0;0;0],
[1;0;0;0],[1;0;0;0],[1;0;0;0],[1;0;0;0],[1;0;0;0],[1;0;0;0],[1;0;0;0],
[1;0;0;0],[1;0;0;0],[1;0;0;0],[1;0;0;0],[1;0;0;0],[1;0;0;0],[1;0;0;0]]

Отже, враховуючи надані ШНМ результати, очевидно, що ідентифіковано дванадцять дефектів з 2-м ступенем критичності їх наслідків (помірних) та тридцять сім дефектів з 1-м ступенем критичності їх наслідків (незначних).

Оскільки, відношення сумарного значення дефектів з 1-м ступенем критичності їх наслідків до загальної кількості виявлених дефектів ПП дорівнює 37/49=0.76, тобто перевищує 0.75, то в аналізованому ПП наявні залишкові дефекти з 2-м ступенем критичності їх наслідків.

Оскільки, відношення сумарного значення дефектів з 2-м ступенем критичності їх наслідків до загальної кількості виявлених дефектів ПП дорівнює 12/49=0.24, тобто не перевищує 0.5, то в аналізованому ПП відсутні залишкові дефекти з 3-м ступенем критичності їх наслідків.

Отже, запропонована інтелектуальна інформаційна технологія ідентифікації залишкових дефектів у ПП надає висновок про наявність залишкових дефектів з 2-м ступенем критичності їх наслідків у програмній частині підсистеми керування освітленням кіберфізичної системи «Розумний будинок» (рисунок 4.13).

Кількість рядків даних			49
37	12	0	0
0,76	0,24	0	0
1	0	0	0

наявності залишкових дефектів 2-го рівня критичності (помірних дефектів)

Рівень критичності дефектів			
незначні	помірні	серйозні	катастрофічні
1	2	3	4
1	0	0	0

Рисунок 4.13 – Зовнішній вигляд програми з представленням висновків про наявність залишкових дефектів у системі «Розумний будинок»

Під час наступного (четвертого) експерименту тестувальник системи на вхід подав звіт про базове тестування ПЗ кіберфізичної системи «Розумна мийка».

Блок семантичного аналізу звіту та підготовки даних для ШНМ виконав семантичний парсинг звіту, вибираючи з нього інформацію про знайдені під час базового тестування дефекти ПЗ, після чого виконав перетворення цієї інформації з лінгвістичної форми представлення в кількісну форму за допомогою таблиць даних бази знань.

ШНМ опрацювала інформацію про дефекти ПЗ, виявлені під час базового тестування, після чого надала наступну інформацію (розмір масиву 4 × 20):

Data4=[[0;0;1;0],[1;0;0;0],[1;0;0;0],[1;0;0;0],[0;0;1;0],[1;0;0;0],[0;0;0;1],
[0;1;0;0],[0;1;0;0],[0;1;0;0],[1;0;0;0],[0;0;0;1],[0;0;1;0],[1;0;0;0],
[1;0;0;0],[0;0;0;1],[1;0;0;0],[0;1;0;0],[0;1;0;0],[0;0;0;1]]

Після розшифрування цих даних стає зрозумілим, що діагностовано чотири дефекти 4-го рівня критичності (катастрофічних), три дефекти 3-го рівня критичності (серйозних), п’ять дефектів 2-го рівня критичності (помірних) та вісім дефектів 1-го рівня критичності (незначних).

Далі, згідно запропонованого методу розробки бездефектного ПЗ шляхом встановлення наявності залишкових дефектів, формуються множини D = (8, 5, 3, 4) та DN = (0,4, 0,25, 3, 4).

Аналізуються всі правила формування висновку про встановлення наявності залишкових дефектів, і за відомими фактами відшукуються заключення, які з цих фактів слідують:

правило 1 не спрацьовує $dn_1 < r_1$;

правило 2 не спрацьовує $dn_2 < r_2$;

правило 3 спрацьовує $dn_3 > r_3$;

правило 4 спрацьовує $dn_4 > r_4$.

Отже, розроблена система формує висновок про встановлення наявності залишкових дефектів 4-го рівня критичності (катастрофічних дефектів) та про непридатність ПЗ «Розумна мийка» і можливу відмову системи (рисунок 4.14).

Кількість рядків даних			20
8	5	3	4
0,40	0,25	3	4
0	0	1	1
непридатність ПЗ (можлива відмова системи) та наявності залишкових дефектів 4-го рівня критичності (катастрофічних дефектів)			
Рівень критичності дефектів			
незначні	помірні	серйозні	катастрофічні
1	2	3	4
0	0	1	0

Рисунок 4.14 – Зовнішній вигляд програми з представленням висновків про наявність залишкових дефектів у системі «Розумна мийка»

Під час п'ятого експерименту тестувальник системи на вхід подав звіт про базове тестування веб-орієнтованої програмної системи діяльності флориста-декоратора «Флорист». Блок семантичного аналізу звіту та підготовки даних для ШНМ виконав семантичний парсинг звіту, вибираючи з нього інформацію про знайдені під час базового тестування дефекти ПЗ, після чого виконав перетворення цієї інформації з лінгвістичної форми представлення в кількісну форму за допомогою таблиць даних бази знань. ШНМ опрацювала інформацію про дефекти ПЗ, виявлені під час базового тестування, після чого надала наступну інформацію (розмір масиву 4 × 30):

Data5=[[1;0;0;0],[1;0;0;0],[1;0;0;0],[1;0;0;0],[1;0;0;0],[1;0;0;0],[1;0;0;0],
[1;0;0;0],[1;0;0;0],[1;0;0;0],[0;0;1;0],[1;0;0;0],[1;0;0;0],[0;1;0;0],
[0;1;0;0],[1;0;0;0],[1;0;0;0],[1;0;0;0],[1;0;0;0],[0;1;0;0],[1;0;0;0],
[1;0;0;0],[1;0;0;0],[1;0;0;0],[1;0;0;0],[1;0;0;0],[1;0;0;0],[0;1;0;0],
[0;1;0;0],[1;0;0;0]]

Після розшифрування цих даних стає зрозумілим, що діагностовано один дефект 3-го рівня критичності (серйозний), п'ять дефектів 2-го рівня критичності (помірних) та двадцять чотири дефекти 1-го рівня критичності (незначних). Далі, згідно запропонованого методу розробки бездефектного ПЗ шляхом встановлення

наявності залишкових дефектів, формуються множини $D = (24, 5, 1, 0)$ та $DN = (0{,}8, 0{,}17, 1, 0)$. Аналізуються всі правила формування висновку про встановлення наявності залишкових дефектів, і за відомими фактами відшукуються заключення, які з цих фактів слідують:

правило 1 спрацьовує $dn_1 > r_1$;

правило 2 не спрацьовує $dn_2 < r_2$;

правило 3 не спрацьовує $dn_3 < r_3$;

правило 4 не спрацьовує $dn_4 < r_4$.

Отже, розроблена система формує висновок про встановлення наявності залишкових дефектів 2-го рівня критичності (помірних дефектів) у системі «Флорист» (рисунок 4.15).

Кількість рядків даних			30
24	5	1	0
0,80	0,17	1	0
1	0	0	0

наявності залишкових дефектів 2-го рівня критичності (помірних дефектів)

Рівень критичності дефектів			
незначні	помірні	серйозні	катастрофічні
1	2	3	4
1	0	0	0

Рисунок 4.15 – Зовнішній вигляд програми з представленням висновків про наявність залишкових дефектів у системі «Флорист»

Під час останнього, шостого експерименту тестувальник системи на вхід подав звіт про базове тестування веб-сервісу для обговорення фільмів «Фільм». Блок семантичного аналізу звіту та підготовки даних для ШНМ виконав семантичний парсинг звіту, вибираючи з нього інформацію про знайдені під час базового тестування дефекти ПЗ, після чого виконав перетворення цієї інформації з лінгвістичної форми представлення в кількісну форму за допомогою таблиць

даних бази знань. ШНМ опрацювала інформацію про дефекти ПЗ, виявлені під час базового тестування, після чого надала інформацію (розмір масиву 4 × 15):

Data6=[[0;1;0;0],[1;0;0;0],[0;1;0;0],[0;1;0;0],[0;1;0;0],[1;0;0;0],[0;1;0;0],
[0;1;0;0],[1;0;0;0],[1;0;0;0],[0;1;0;0],[1;0;0;0],[1;0;0;0],[0;1;0;0],[0;1;0;0]]

Після розшифрування цих даних стає зрозумілим, що діагностовано дев'ять дефектів 2-го рівня критичності (помірних) та шість дефектів 1-го рівня критичності (незначних). Далі, згідно розробленого методу розробки бездефектного ПЗ шляхом встановлення наявності залишкових дефектів, формуються множини $D = (6, 9, 0, 0)$ та $DN = (0{,}4, 0{,}6, 0, 0)$. Аналізуються усі правила формування висновку про встановлення наявності залишкових дефектів, і за відомими фактами відшукуються заключення, які з цих фактів слідують:

правило 1 не спрацьовує $dn_1 < r_1$;

правило 2 спрацьовує $dn_2 > r_2$;

правило 3 не спрацьовує $dn_3 < r_3$;

правило 4 не спрацьовує $dn_4 < r_4$.

Отже, розроблена система формує висновок про встановлення наявності залишкових дефектів 3-го рівня критичності (серйозних дефектів) у системі «Фільм» (рисунок 4.16).

Кількість рядків даних			15
6	9	0	0
0,40	0,60	0	0
0	1	0	0
наявності залишкових дефектів 3-го рівня критичності (серйозних дефектів)			
Рівень критичності дефектів			
незначні	помірні	серйозні	катастрофічні
1	2	3	4
0	1	0	0

Рисунок 4.16 – Зовнішній вигляд програми з представленням висновків про наявність залишкових дефектів у системі «Фільм»

Шляхом збору підсумкової інформації, програма також дозволяє будувати графіки та порівнювати між собою результати проведених експериментів (рисунок 4.17).

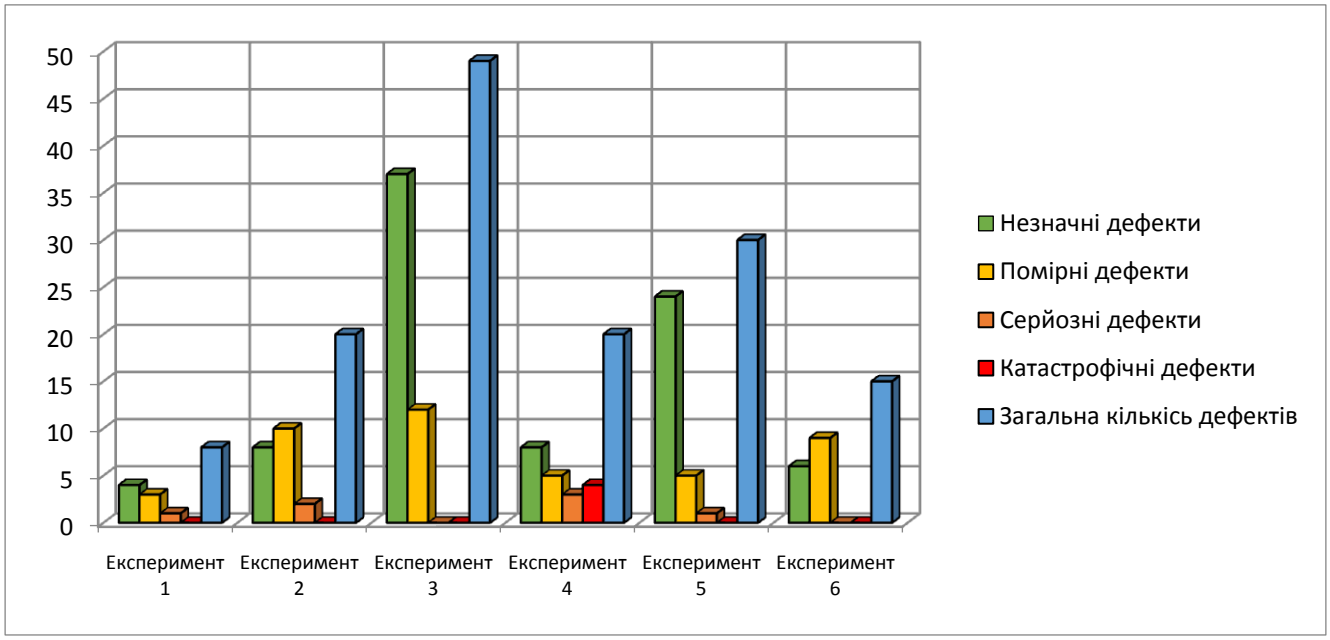


Рисунок 4.17 – Діаграма виявлених дефектів ПЗ різних рівнів критичності у результаті 6-х проведених експериментів

Окремо було проведено експерименти застосунків для мобільних телефонів, написаних на мові програмування Kotlin. ПП, які підлягали дослідженню були мобільні застосунки (МЗ): «Нове таксі», «Інтерактивна карта», «Бібліотека» та «Аптека».

Під час першого експерименту тестувальник подав звіт про дефекти у МЗ «Нове таксі». Блок семантичного аналізу звіту та підготовки даних для ШНМ виконав парсинг поданого звіту, обираючи з нього інформацію про знайдені під час базового тестування дефекти МЗ та виконав перетворення цієї інформації з лінгвістичної форми представлення в кількісну форму. ШНМ опрацювала інформацію про дефекти ПП, виявлені під час базового тестування, після створивши масив розміром 4 × 31):

Data7=[[0;1;0;0],[1;0;0;0],[1;0;0;0],[1;0;0;0],[1;0;0;0],[1;0;0;0],[0;1;0;0],
[0;1;0;0],[1;0;0;0],[1;0;0;0],[0;1;0;0],[1;0;0;0],[1;0;0;0],[0;1;0;0],

[0;1;0;0],[0;1;0;0],[1;0;0;0],[1;0;0;0],[0;1;0;0],[1;0;0;0],[1;0;0;0],
[0;1;0;0],[0;1;0;0],[0;1;0;0],[1;0;0;0],[1;0;0;0],[0;1;0;0],[1;0;0;0],
[1;0;0;0],[1;0;0;0],[1;0;0;0]]

Після розшифрування цих даних стає зрозумілим, що діагностовано дванадцять дефектів 2-го рівня критичності (помірних) та дев'ятнадцять дефектів 1-го рівня критичності (незначних).

Далі, згідно розробленого методу розробки бездефектного МЗ, шляхом встановлення наявності залишкових дефектів, формуються множини $D = (19, 12, 0, 0)$ та $DN = (0{,}61, 0{,}39, 0, 0)$. Аналізуються усі правила формування висновку про встановлення наявності залишкових дефектів, і за відомими фактами відшукуються заключення, які з цих фактів слідують:

правило 1 не спрацьовує $dn_1 < r_1$;

правило 2 не спрацьовує $dn_2 < r_2$;

правило 3 не спрацьовує $dn_3 < r_3$;

правило 4 не спрацьовує $dn_4 < r_4$.

Отже, розроблена система формує висновок про відсутність залишкових дефектів у МЗ «Нове таксі» (рисунок 4.18).

Кількість рядків даних			31
19	12	0	0
0,61	0,39	0	0
0	0	0	0

відсутність залишкових дефектів

Рівень критичності дефектів			
незначні	помірні	серйозні	катастрофічні
1	2	3	4
0	1	0	0

Рисунок 4.18 – Зовнішній вигляд програми з представленням висновків про наявність залишкових дефектів у МЗ «Нове таксі»

Під час другого експерименту тестувальник подав звіт про тестування МЗ «Інтерактивна карта». Блок семантичного аналізу звіту та підготовки даних для ШНМ виконав парсинг поданого звіту, обираючи з нього інформацію про знайдені під час базового тестування дефекти МЗ та виконав перетворення цієї інформації з лінгвістичної форми представлення в кількісну форму.

ШНМ опрацювала інформацію про дефекти МЗ, виявлені під час базового тестування, після створивши масив розміром 4 × 40):

Data8=[[0;1;0;0],[1;0;0;0],[1;0;0;0],[1;0;0;0],[1;0;0;0],[1;0;0;0],[0;1;0;0],
[0;1;0;0],[1;0;0;0],[1;0;0;0],[0;1;0;0],[1;0;0;0],[1;0;0;0],[0;1;0;0],
[0;1;0;0],[0;1;0;0],[1;0;0;0],[1;0;0;0],[0;1;0;0],[1;0;0;0],[1;0;0;0],
[0;1;0;0],[0;1;0;0],[0;1;0;0],[1;0;0;0],[1;0;0;0],[0;1;0;0],[1;0;0;0],
[0;1;0;0],[0;1;0;0],[1;0;0;0],[1;0;0;0],[0;1;0;0],[1;0;0;0],[1;0;0;0],
[0;1;0;0],[0;1;0;0],[0;1;0;0],[0;1;0;0],[1;0;0;0]]

Тобто, діагностовано дев'ятнадцять дефектів 2-го рівня критичності (помірних) та двадцять один дефект 1-го рівня критичності (незначних).

Далі, згідно розробленого методу розробки бездефектного МЗ шляхом встановлення наявності залишкових дефектів, формуються множини D = (21, 19, 0, 0) та DN = (0,53, 0,48, 0, 0).

Аналізуються усі правила формування висновку про встановлення наявності залишкових дефектів, і за відомими фактами відшукуються заключення, які з цих фактів слідують:

правило 1 не спрацьовує $dn_1 < r_1$;

правило 2 не спрацьовує $dn_2 < r_2$;

правило 3 не спрацьовує $dn_3 < r_3$;

правило 4 не спрацьовує $dn_4 < r_4$.

Отже, розроблена система формує висновок про відсутність залишкових дефектів у МЗ «Інтерактивна карта» (рисунок 4.19).

Кількість рядків даних			40
21	19	0	0
0,53	0,48	0	0
0	0	0	0
відсутність залишкових дефектів			
Рівень критичності дефектів			
незначні	помірні	серйозні	катастрофічні
1	2	3	4
0	1	0	0

Рисунок 4.19 – Зовнішній вигляд програми з представленням висновків про наявність залишкових дефектів у МЗ «Інтерактивна карта»

Для третього експерименту тестувальник подав звіт про тестування МЗ «Бібліотека». Блок семантичного аналізу звіту та підготовки даних для ШНМ виконав парсинг поданого звіту, обираючи з нього інформацію про знайдені під час базового тестування дефекти МЗ та виконав перетворення цієї інформації з лінгвістичної форми представлення в кількісну форму.

ШНМ опрацювала інформацію про дефекти МЗ, виявлені під час базового тестування, після створивши масив розміром 4 × 25):

Data9=[[0;1;0;0],[1;0;0;0],[1;0;0;0],[1;0;0;0],[1;0;0;0],[1;0;0;0],[0;1;0;0],

[0;1;0;0],[1;0;0;0],[1;0;0;0],[0;1;0;0],[0;0;1;0],[1;0;0;0],[0;1;0;0],

[0;1;0;0],[0;1;0;0],[1;0;0;0],[1;0;0;0],[0;1;0;0],[1;0;0;0],[1;0;0;0],

[0;1;0;0],[0;1;0;0],[0;1;0;0],[1;0;0;0]]

Тобто, діагностовано один дефект 3-го рівня критичності (серйозних), одинадцять дефектів 2-го рівня критичності (помірних) та тринадцять дефектів 1-го рівня критичності (незначних).

Далі, згідно розробленого методу розробки бездефектного МЗ шляхом встановлення наявності залишкових дефектів, формуються множини D = (13, 11, 1, 0) та DN = (0,52, 0,44, 1, 0).

Аналізуються усі правила формування висновку про встановлення наявності залишкових дефектів, і за відомими фактами відшукуються заключення, які з цих фактів слідують:

правило 1 не спрацьовує $dn_1 < r_1$;

правило 2 не спрацьовує $dn_2 < r_2$;

правило 3 не спрацьовує $dn_3 < r_3$;

правило 4 не спрацьовує $dn_4 < r_4$.

Отже, розроблена система формує висновок про відсутність залишкових дефектів у МЗ «Бібліотека» (рисунок 4.20).

Кількість рядків даних			25
13	11	1	0
0,52	0,44	1	0
0	0	0	0

відсутність залишкових дефектів

Рівень критичності дефектів			
незначні	помірні	серйозні	катастрофічні
1	2	3	4
0	1	0	0

Рисунок 4.20 – Зовнішній вигляд програми з представленням висновків про наявність залишкових дефектів у МЗ «Бібліотека»

Для четвертого експерименту було подано звіт про базове тестування МЗ «Аптека». Блок семантичного аналізу звіту та підготовки даних для ШНМ виконав семантичний парсинг звіту, вибираючи з нього інформацію про знайдені під час базового тестування дефекти МЗ, після чого виконав перетворення цієї інформації з лінгвістичної форми представлення в кількісну форму за допомогою таблиць даних бази знань. ШНМ опрацювала інформацію про дефекти МЗ, виявлені під час базового тестування, після чого надала наступну інформацію (розмір масиву 4 × 20):

Data10=[[0;0;1;0],[1;0;0;0],[1;0;0;0],[1;0;0;0],[0;0;1;0],[1;0;0;0],[0;1;0;0],

[0;1;0;0],[0;1;0;0],[0;1;0;0],[1;0;0;0],[0;1;0;0],[0;0;1;0],[1;0;0;0],
[1;0;0;0],[0;1;0;0],[1;0;0;0],[0;1;0;0],[0;0;1;0],[0;1;0;0]]

Після розшифрування цих даних стає зрозумілим, що діагностовано чотири дефекти 3-го рівня критичності (серйозних), вісім дефектів 2-го рівня критичності (помірних) та вісім дефектів 1-го рівня критичності (незначних).

Далі, згідно запропонованого методу розробки бездефектного ПЗ шляхом встановлення наявності залишкових дефектів, формуються множини $D = (8, 8, 4, 0)$ та $DN = (0{,}4, 0{,}4, 4, 0)$. Аналізуються всі правила формування висновку про встановлення наявності залишкових дефектів, і за відомими фактами відшукуються заключення, які з цих фактів слідують:

правило 1 не спрацьовує $dn_1 < r_1$;

правило 2 не спрацьовує $dn_2 < r_2$;

правило 3 спрацьовує $dn_3 > r_3$;

правило 4 не спрацьовує $dn_4 < r_4$.

Отже, розроблена система формує висновок про встановлення наявності залишкових дефектів 4-го рівня критичності (катастрофічних дефектів) у МЗ «Аптека» (рисунок 4.21).

Кількість рядків даних			20
8	8	4	0
0,40	0,40	4	0
0	0	1	0

наявності залишкових дефектів 4-го рівня критичності (катастрофічних дефектів)

Рівень критичності дефектів			
незначні	помірні	серйозні	катастрофічні
1	2	3	4
0	0	1	0

Рисунок 4.21 – Зовнішній вигляд програми з представленням висновків про наявність залишкових дефектів МЗ «Аптека»

Шляхом збору підсумкової інформації про кількість дефектів у МЗ, була побудована діаграма (рисунок 4.22).

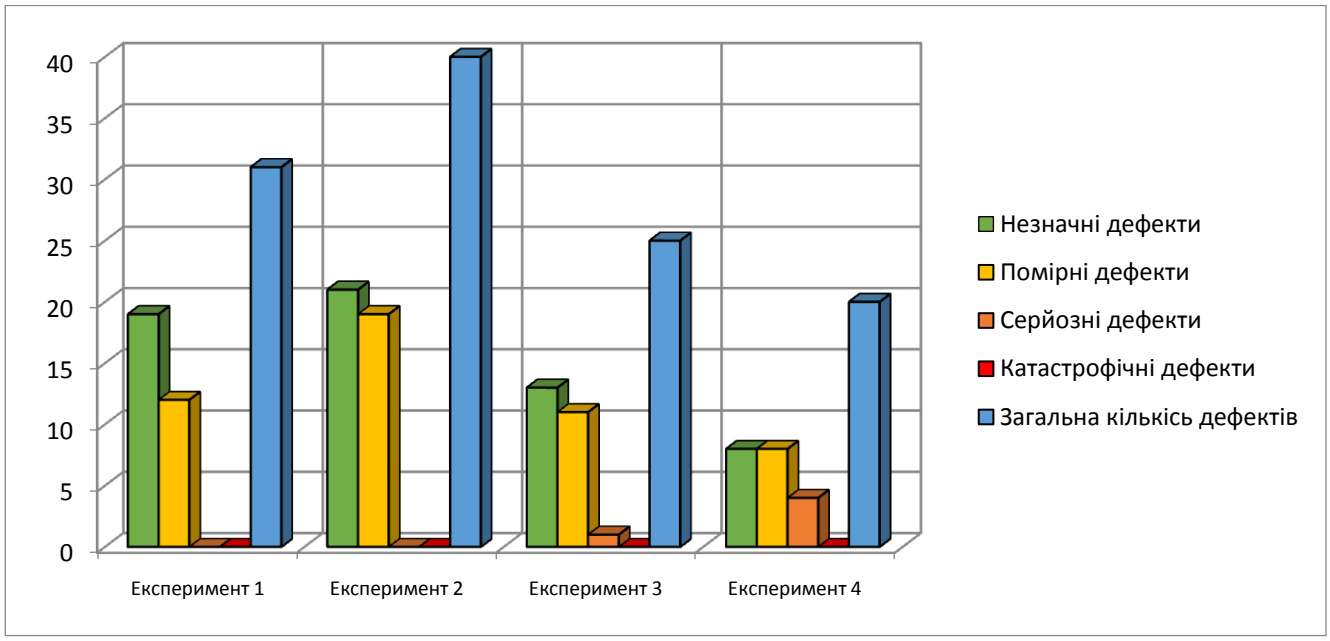


Рисунок 4.22 – Діаграма виявлених дефектів у МЗ різних рівнів критичності у результаті 4-х проведених експериментів

Як показали проведені експерименти, запропонована система розробки бездефектного ПЗ шляхом встановлення наявності залишкових дефектів в результаті свого функціонування видає висновок про встановлення наявності або відсутності залишкових дефектів (із зазначенням рівня(ів) критичності залишкових дефектів у разі встановлення їх наявності) на основі опрацювання інформації про кількість і типи дефектів, виявлених під час базового тестування, що міститься у звіті базового тестування.

Тому, запропонована програма дозволяє розробляти бездефектне ПЗ виявлення залишкових дефектів у ПЗ після базового тестування, за рахунок чого відбувається підвищення достовірності тестування і відповідно зростання якості ПЗ.

4.4 Висновки

У четвертому розділі запропонована інтелектуальна інформаційна технологія ідентифікації залишкових дефектів у ПЗ, яка забезпечує висновок про відсутність залишкових дефектів або про наявність залишкових дефектів з різними ступенями критичності їх наслідків на основі природомовного звіту про дефекти ПЗ, виявлені під час його базового тестування.

Як інструмент запропонованої інформаційної технології запропонована система розробки бездефектного ПЗ шляхом встановлення наявності залишкових дефектів, яка дозволяє користувачу, на основі звіту про результати основного тестування, одержати висновок про відсутність чи наявність залишкових дефектів у програмному забезпеченні (із зазначенням рівня(ів) критичності залишкових дефектів у разі встановлення їх наявності).

У четвертому розділі також апробовано розроблену інформаційну технологію ідентифікації залишкових дефектів у ПП з проведенням десяти експериментів.

Для першого експерименту запропонована інтелектуальна інформаційна технологія ідентифікації залишкових дефектів надала висновок про відсутність залишкових дефектів у ПП «Ресторан», оскільки жодне із правил не спрацювало.

У другому експерименті запропонована інтелектуальна інформаційна технологія ідентифікації залишкових дефектів у ПП надала висновок про наявність залишкових дефектів з 3-м та 4-м ступенями критичності їх наслідків у програмній частині кіберфізичної системи «Розумна парковка».

Під час проведення третього експерименту, запропонована інтелектуальна інформаційна технологія ідентифікації залишкових дефектів у ПП надала висновок про наявність залишкових дефектів з 2-м ступенем критичності їх наслідків у програмній частині підсистеми керування освітленням кіберфізичної системи «Розумний будинок».

У четвертому експерименті розроблена система сформувала висновок про встановлення наявності залишкових дефектів 4-го рівня критичності

(катастрофічних дефектів) та про непридатність ПЗ і можливу відмову системи «Розумна мийка».

Під час проведення п'ятого експерименту, розроблена система сформувала висновок про встановлення наявності залишкових дефектів 2-го рівня критичності (помірних дефектів) у системі «Флорист».

В останньому шостому експерименті розроблена система сформувала висновок про встановлення наявності залишкових дефектів 3-го рівня критичності (серйозних дефектів) у системі «Фільм». Як показали проведені дослідження, розроблена система розробки бездефектного ПЗ шляхом встановлення наявності залишкових дефектів в результаті свого функціонування видає висновок про встановлення наявності або відсутності залишкових дефектів (із зазначенням рівня(ів) критичності залишкових дефектів у разі встановлення їх наявності) на основі опрацювання інформації про кількість і типи дефектів, виявлених під час базового тестування, що міститься у звіті базового тестування. Тому, запропонована програма дозволяє розробляти бездефектне ПЗ виявлення залишкових дефектів у програмах після базового тестування, за рахунок чого відбувається підвищення достовірності тестування і відповідно зростання якості ПЗ.

Окремо було проведено експерименти застосунків для мобільних телефонів, написаних на мові програмування Kotlin. ПП, які підлягали дослідженню були мобільні застосунки (МЗ): «Нове таксі», «Інтерактивна карта», «Бібліотека» та «Аптека».

Для першого, другого і третього експериментів запропонована інтелектуальна інформаційна технологія ідентифікації залишкових дефектів надала висновок про відсутність залишкових дефектів у МЗ «Нове таксі», «Інтерактивна карта» і «Бібліотека».

Для четвертого експерименту розроблена система сформувала висновок про встановлення наявності залишкових дефектів 4-го рівня критичності (катастрофічних дефектів) у МЗ «Аптека».

ВИСНОВКИ

За результатами виконаних теоретичних та практичних досліджень у роботі розроблено метод та засоби інформаційної технології ідентифікації залишкових дефектів у ПЗ.

У першому розділі проаналізовано існуючі моделі, методи та засоби ідентифікації залишкових дефектів у ПЗ. Виявлено, що до основних моделей для ідентифікації залишкових дефектів відносяться наступні: машинне навчання; статичний аналіз; динамічний аналіз; комбінований аналіз. Також проведений аналіз дозволив виявити тісний взаємозв'язок обраних методів з життєвим циклом дефекту. Це дозволило виявити методи для ідентифікації залишкових дефектів а саме: статичні та динамічні; рецензування коду; автоматизоване тестування; безперервну інтеграцію та розгортання; контроль версій; рефакторинг. У розділі проаналізовано засоби (системи автоматизованого тестування, моніторинг та журналювання, методи аналізу коду та інші), які можливо використовувати при ідентифікації залишкових дефектів у ПЗ на різних стадіях розробки ПП. В роботі пропонується використовувати автоматизовані засоби тестування.

У другому розділі доведено, що зростання щільності дефектів залежить від кількості рядків програмного коду. Визначено типову щільність дефектів на 1000 рядків коду для ПЗ різного розміру. Встановлено, що мінімальна щільність дефектів при кількості коду менше двох тисяч складає – 1, а при більше 512 тисяч – 5. Відповідно, максимальна щільність дефектів при кількості коду менше двох тисяч складає – 22, а при більше 512 тисяч – 98.

З метою ідентифікації у ПЗ залишкових дефектів з різними ступенями критичності їх наслідків розглянуто інформацію про дефекти, а саме про їх кількість і типи, які були виявлені під час його тестування. Для отримання висновку про відсутність або наявність залишкових дефектів з різними ступенями критичності їх наслідків, в роботі рекомендовано враховувати взаємний вплив виявлених та залишкових дефектів ПЗ та інші фактори. Для розв'язання такої

задачі запропоновано використовувати ШНМ. Для цього розроблено концепцію інформаційної технології ідентифікації залишкових дефектів у ПЗ.

Доведено, якщо відношення сумарного значення дефектів 1-го рівня критичності (незначних дефектів) до загальної кількості виявлених під час базового тестування дефектів дорівнює або перевищує 0.75, то приймається висновок про встановлення наявності залишкових дефектів 2-го рівня критичності (помірних дефектів). Якщо відношення сумарного значення дефектів 2-го рівня критичності (помірних дефектів) до загальної кількості виявлених під час базового тестування дефектів дорівнює або перевищує 0.5, то приймається висновок про встановлення наявності залишкових дефектів 3-го рівня критичності (серйозних дефектів). Якщо кількість дефектів 3-го рівня критичності (серйозних) перевищує 2, то приймається висновок про встановлення наявності залишкових дефектів 4-го рівня критичності (катастрофічних дефектів). Якщо кількість дефектів 4-го рівня критичності (катастрофічних) перевищує або дорівнює 1, то приймається висновок про непридатність програмного забезпечення і можливу відмову системи. Отже, встановлено, що пороги мають наступні значення: 0.75; 0.5; 2; 1. При формуванні бази знань запропоновано створити її з розділу даних і розділу правил.

У третьому розділі вдосконалено нейромережну модель ідентифікації залишкових дефектів у програмному забезпеченні на основі звіту основного тестування, яка відрізняється від відомих тим, що дає можливість враховувати важливість кожного рядка звіту основного тестування, а також взаємний вплив атрибутів в межах дефекту кожного рівня критичності. Вихідні функціонали ШНМ, що відповідають значенням рівнів критичності, дають можливість оцінити сумарний вплив відшуканих під час основного тестування помилок на наявність залишкових дефектів у програмному забезпеченні.

Крім цього, розроблено метод ідентифікації залишкових дефектів у програмному забезпеченні, суть якого полягає у виявленні множини дефектів різних рівнів критичності та аналізу цієї множини на предмет наявності або відсутності залишкових дефектів. Метод відрізняється від відомих тим, що вхідна

інформація про результати основного тестування опрацьовується штучною нейронною мережею.

В третьому розділі також розроблено вимоги до інформаційної технології ідентифікації залишкових дефектів у програмному забезпеченні.

У четвертому розділі запропонована інтелектуальна інформаційна технологія ідентифікації залишкових дефектів у ПЗ, яка забезпечує висновок про відсутність залишкових дефектів або про наявність залишкових дефектів з різними ступенями критичності їх наслідків на основі природомовного звіту про дефекти ПЗ, виявлені під час його базового тестування.

Як інструмент запропонованої інформаційної технології запропонована система розробки бездефектного ПЗ шляхом встановлення наявності залишкових дефектів, яка дозволяє користувачу, на основі звіту про результати основного тестування, одержати висновок про відсутність чи наявність залишкових дефектів у програмному забезпеченні (із зазначенням рівня(ів) критичності залишкових дефектів у разі встановлення їх наявності).

У четвертому розділі також апробовано розроблену інформаційну технологію ідентифікації залишкових дефектів у ПП з проведенням десяти експериментів. Для першого експерименту запропонована інтелектуальна інформаційна технологія ідентифікації залишкових дефектів надала висновок про відсутність залишкових дефектів у ПП «Ресторан», оскільки жодне із правил не спрацювало.

У другому експерименті запропонована інтелектуальна інформаційна технологія ідентифікації залишкових дефектів у ПП надала висновок про наявність залишкових дефектів з 3-м та 4-м ступенями критичності їх наслідків у програмній частині кіберфізичної системи «Розумна парковка».

Під час проведення третього експерименту, запропонована інтелектуальна інформаційна технологія ідентифікації залишкових дефектів у ПП надала висновок про наявність залишкових дефектів з 2-м ступенем критичності їх наслідків у програмній частині підсистеми керування освітленням кіберфізичної системи «Розумний будинок».

У четвертому експерименті розроблена система сформувала висновок про встановлення наявності залишкових дефектів 4-го рівня критичності (катастрофічних дефектів) та про непридатність ПЗ і можливу відмову системи «Розумна мийка».

Під час проведення п'ятого експерименту, розроблена система сформувала висновок про встановлення наявності залишкових дефектів 2-го рівня критичності (помірних дефектів) у системі «Флорист».

В останньому, шостому, експерименті розроблена система сформувала висновок про встановлення наявності залишкових дефектів 3-го рівня критичності (серйозних дефектів) у системі «Фільм».

Окремо було проведено експерименти застосунків для мобільних телефонів, написаних на мові програмування Kotlin. ПП, які підлягали дослідженню були мобільні застосунки (МЗ): «Нове таксі», «Інтерактивна карта», «Бібліотека» та «Аптека».

Для першого, другого і третього експериментів запропонована інтелектуальна інформаційна технологія ідентифікації залишкових дефектів надала висновок про відсутність залишкових дефектів у МЗ «Нове таксі», «Інтерактивна карта» і «Бібліотека».

Для четвертого експерименту розроблена система сформувала висновок про встановлення наявності залишкових дефектів 4-го рівня критичності (катастрофічних дефектів) у МЗ «Аптека».

Як показали проведені дослідження, розроблена система розробки бездефектного ПЗ шляхом встановлення наявності залишкових дефектів в результаті свого функціонування видає висновок про встановлення наявності або відсутності залишкових дефектів (із зазначенням рівня(ів) критичності залишкових дефектів у разі встановлення їх наявності) на основі опрацювання інформації про кількість і типи дефектів, виявлених під час базового тестування, що міститься у звіті базового тестування.

Тому, запропонована програма дозволяє розробляти бездефектне ПЗ виявлення залишкових дефектів у програмах після базового тестування, за

рахунок чого відбувається підвищення достовірності тестування і відповідно зростання якості ПЗ.

Наукова новизна отриманих результатів:

– набув подальшого розвитку метод ідентифікації залишкових дефектів у програмному забезпеченні, який відрізняється від відомих тим, що вхідна інформація про результати основного тестування обробляється штучною нейронною мережею (ШНМ), і забезпечує визначення множини дефектів різного рівня критичності та аналіз цієї множини на наявність або відсутність залишкових дефектів;

– набула подальшого розвитку інформаційна технологія ідентифікації залишкових дефектів у програмних продуктах за рахунок опрацювання природомовних звітів про результати основного тестування, яка забезпечує висновок про відсутність або наявність залишкових дефектів у аналізованому програмному забезпеченні.

Практична значущість отриманих результатів полягає у проєктуванні та реалізації інтелектуальної інформаційної технології ідентифікації залишкових дефектів у програмних продуктах, яка забезпечує висновок про відсутність залишкових дефектів або про наявність залишкових дефектів з різними ступенями критичності їх наслідків на основі природомовного звіту про дефекти програмного продукту, виявлені під час його основного тестування.

За темою кваліфікаційної роботи опубліковано одну статтю у матеріалах конференції, що індексуються в наукометричній базі Scopus, і одну статтю у фаховому виданні України:

1) I. Zasornova, T. Hovorushchenko, M. Fedula, V. Buzyl. Intelligent Information Technology for Identification of Remaining Defects in Software Products // Proceedings of the 2023 IEEE 12-th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS-2023, Dortmund, 7-9 September 2023), vol. 1, pp. 33-37;

2) I. Zasornova, T. Hovorushchenko, O. Voichur. Study of Software Testing Tools According to the Testing Levels. Computer Systems & Information Technologies. 2023. №1. Pp. 38-46.

Взято участь у The 12th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS). (September 7-9, 2023, Dortmund, Germany).

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Zasornova I. Intelligent Information Technology for Identification of Remaining Defects in Software Products / I. Zasornova, T. Hovorushchenko, M. Fedula, V. Buzyl // IDAACS. 2023. Pp. 33-37.

2. Zasornova I.O. Study of software testing tools according to the testing levels / I.O. Zasornova, T.O. Hovorushchenko, O.Y. Voichur // Computer Systems & Information Technologies. 2023. №1. Pp. 38-46.

3. L. Li, and Y. Tian, Application of Data Guidance Site Generation Technology in the Cloud Platform Supporting the Construction of Subject Teams in Finance and Economics Applied Universities, 2022 International Conference on Edge Computing and Applications, India, pp. 19-22, November 2022.

4. K. Król, and D. Zdonek, Local Government Website Accessibility – Evidence from Poland Administrative sciences, vol. 10, paper №22, 2020.

5. K. Kirk, and A. S. Abrahams, Evaluating public charity websites. Stage Model versus Automated Service, Nonprofit Management & Leadership, pp. 475-491, 2017.

6. Y. Akgül, Web Accessibility Evaluation of Government Websites for People with Disabilities in Turkey, Journal of Advanced Management Science, vol. 4, №3, pp. 201-210, 2016.

7. K. Fatima, N.Z. Bawany, and M. Bukhari, Usability and Accessibility Evaluation of Banking Websites, 2020 International Conference on Advanced Computer Science and Information Systems, Indonesian, pp. 247-256, November 2020.

8. T. Hovorushchenko, O. Pavlova, and D. Medzatyi, Ontology-Based Intelligent Agent for Determination of Sufficiency of Metric Information in the Software Requirements, Advances in Intelligent Systems and Computing, vol. 1020, pp. 447-460, 2020.

9. M. Rennhard, M. Kushnir, O. Favre, D. Esposito, and V. Zahnd, Automating the Detection of Access Control Vulnerabilities in Web Applications, SN Computer Science, vol. 3, paper №376, 2022.

10. H.-P. Nguyen, T.-N. Luong, and N.-T. Truong, Generating Test Paths to Detect XSS Vulnerabilities of Web Applications, 2022 9th NAFOSTED Conference on Information and Computer Science, Vietnam, pp. 287-293, 2022.

11. M. Farokhad, J. Otegi-Olaso, L. Pinilla, N. Gandarias, and L. de Lacalle, Assessing the Success of R&D Projects and Innovation Projects through Project Management Life Cycle, 2019 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), France, pp. 1104-1110, September 2019.

12. M. Surendra Naidu et al. Classification of defects in software using decision tree algorithm. International Journal of Engineering Science and Technology (IJEST). Vol. 5 №06, June 2013.

13. R. Natella, S. Winter, D. Cotroneo and N. Suri, Analyzing the Effects of Bugs on Software Interfaces, IEEE Transactions on Software Engineering, vol. 46, №3, pp. 280-301, 2020.

14. E. Zaitseva, V. Levashenko, J. Rabcan, M. Kvassay, and P. Rusnak, Reliability Evaluation of Multi-State System Based on Incompletely Specified Data and Structure Function, 2019 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), France, pp. 741-746, September 2019.

15. Duhan, M., & Bhatia, P. K. (2022). Software Reusability Estimation based on Dynamic Metrics using Soft Computing Techniques. International Journal of Computing, 21(2), 188-194. https://doi.org/10.47839/ijc.21.2.2587.

16. How tech firm Shadow sought to revolutionize Democratic campaigns – but stumbled in Iowa. [Online]. Available: https://www.washingtonpost.com/technology/2020/02/04/how-tech-firm-shadow-sought-revolutionize-democratic-campaigns-stumbled-iowa. [Accessed 9 March 2023].

17. British Airways passengers stranded after IT failures. [Online]. Available: https://www.bbc.com/news/uk-49261497. [Accessed 9 March 2023].

18. Starliner anomaly to prevent ISS docking. [Online]. Available: https://spacenews.com/starliner-anomaly-to-prevent-iss-docking. [Accessed 9 March 2023].

19. The 2018 Software Fail Watch Awards. [Online]. Available: https://www.tricentis.com/blog/software-fail-awards-2018. [Accessed 9 March 2023].

20. Fiat Chrysler is recalling more than 1.25 million pickup trucks worldwide over a software error that «may be related» to a death and two injuries. [Online]. Available: https://www.bbc.com/news/technology-39898319. [Accessed 9 March 2023].

21. What is the cost of poor software quality in the U.S.? [Online]. Available: https://www.synopsys.com/blogs/software-security/poor-software-quality-costs-us. [Accessed 9 March 2023].

22. Software Fails Watch. [Online]. Available: https://www.tricentis.com/wp-content/uploads/2019/01/Software-Fails-Watch-5th-edition.pdf. [Accessed 9 March 2023].

23. What is the actual cost of software failures? [Online]. Available: https://undo.io/the-cost-of-software-failures. [Accessed 9 March 2023].

24 Liu, L., Zhao, J., Chen, Z., Zhao, B., & Ji, Y. A New Bolt Defect Identification Method Incorporating Attention Mechanism and Wide Residual Networks. Sensors, vol. 22, №19, 2022, p. 7416. doi:10.3390/s22197416.

25. Zhu, A., Ma, C., Chen, S., Wang, B., & Guo, H. Tunnel Lining Defect Identification Method Based on Small Sample Learning. Wireless Communications and Mobile Computing, vol. 2022, 2022, pp. 1–9. doi:10.1155/2022/1096467.

26 Foss, K., Couckuyt, I., Baruta, A., & Mossoux, C. Automated Software Defect Detection and Identification in Vehicular Embedded Systems. IEEE Transactions on Intelligent Transportation Systems, vol. 23, №7, 2022, pp. 6963–6973. doi:10.1109/tits.2021.3065940.

27 Gao, L., & Wong, W. E. An empirical study of software defect prediction with a simplified dependency analysis. IEEE Transactions on Software Engineering, vol. 45, №4, 2019, pp. 373-392. doi:10.1109/tse.2017.2764603.

28 Sun, H., Peng, L., Huang, S., Li, S., Long, Y., Wang, S., & Zhao, W. Development of a Physics-Informed Doubly Fed Cross-Residual Deep Neural Network for High-Precision Magnetic Flux Leakage Defect Size Estimation. IEEE Transactions on Industrial Informatics, vol. 18, №3, 2022, pp. 1629-1640. doi:10.1109/tii.2021.3089333.

29. Zou, Y., & Li, M. A survey on software defect prediction. Journal of Computer Science and Technology, vol. 34, № 1, 2019, pp. 1-24. doi:10.1007/s11390-019-1917-7.

30. Machida, F., & Onoue, A. Dynamic analysis for defect detection in software. In 2019 IEEE 12th International Conference on Software Testing, Verification and Validation (ICST), 2019, pp. 133-140. doi:10.1109/icst.2019.00022.

31. He, Z., Liu, J., & Wang, Z. A survey of dynamic analysis methods for malware detection. In 2019 IEEE 13th International Symposium on Autonomous Decentralized Systems (ISADS), 2019, pp. 112-118. doi:10.1109/isads.2019.8705515.

32. AlOmar, E. A., Mkaouer, M. W., Ouni, A., & Kessentini, M. On the Impact of Refactoring on the Relationship between Quality Attributes and Design Metrics. 2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), September 2019, https://doi.org/10.1109/esem.2019.8870177.

33. Mashhadi, E., Chowdhury, S., Modaberi, S., Hemmati, H., & Uddin, G. An Empirical Study on Bug Severity Estimation Using Source Code Metrics and Static Analysis (Version 1). arXiv, 2022, https://doi.org/10.48550/ARXIV.2206.12927.

34. Mashhadi, E. EhsanMashhadi/ISSRE2023-BugSeverityPrediction: 0.1.0 (0.1.0) [Computer software]. Zenodo, 2023, https://doi.org/10.5281/ZENODO.8267597.

35. H.-P. Nguyen, T.-N. Luong, and N.-T. Truong, Generating Test Paths to Detect XSS Vulnerabilities of Web Applications, 2022 9th NAFOSTED Conference on Information and Computer Science, Vietnam, pp. 287-293, 2022.

36. M. Felderer, P. Zech, R. Breu, M. Büchler, and A. Pretschner, Model-based security testing: a taxonomy and systematic classification, Software Testing, Verifiation & Reliability, vol. 26, issue 2, pp. 119-148, 2016.

37. S. Nair, J. L. de la Vara, M. Sabetzadeh, and L. Briand, Classification, Structuring, and Assessment of Evidence for Safety - A Systematic Literature Review, 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation, Luxembourg, pp. 94-103, March 2013.

38. M. Felderer, B. Agreiter, P. Zech, and R. Breu, A Classification for Model-Based Security Testing, The Third International Conference on Advances in System Testing and Validation Lifecycle, Spain, pp. 109-114, December 2011.

39. T. Hovorushchenko, P. Popov. Method of Developing the Defect-Free Medical Software by Establishing the Presence of Residual Defects. CEUR-WS. 2021. Vol. 3038. Pp. 11-21.

40. O. Drozd, I. Perebeinos, O. Martynyuk, K. Zashcholkin, O. Ivanova, M. Drozd, Hidden fault analysis of FPGA projects for critical applications, in: Proceedings of IEEE International Conference. Modern problems of radio engineering, telecommunications and computer science, TCSET'2020, Lviv-Slavsko, 2020, paper 142. doi: 10.1109/TCSET49122.2020.235591.

41. O. Mishchuk, R. Tkachenko, I. Izonin, Missing data imputation through SGTM neural-like structure for environmental monitoring tasks. Advances in Intelligent Systems and Computing 938 (2019) 142-151. doi: 10.1007/978-3-030-16621-2_13.

42. S. McConnell, Code complete, Microsoft Press, Redmond, 2013.

43. T. Hovorushchenko, Methodology of evaluating the sufficiency of information for software quality assessment according to ISO 25010. Journal of Information and Organizational Sciences 42, №1, (2018), pp. 63-85. doi: 10.31341/jios.42.1.4.

44 T. Hovorushchenko, O. Pavlova, M. Bodnar, Development of an intelligent agent for analysis of nonfunctional characteristics in specifications of software requirements. Eastern-European Journal of Enterprise Technologies 1, №2, (2019), pp. 6-17. doi: 10.15587/1729-4061.2019.154074.

45. T. Hovorushchenko, O. Pavlova, Evaluating the software requirements specifications using ontology-based intelligent agent, in: Proceedings of 2018 IEEE International Scientific and Technical Conference. Computer Science and Information

Technologies, CSIT'2018, Lviv, 2018, vol. 1, pp.215-218. doi: 10.1109/STC-CSIT.2018.8526730.

46. T. Hovorushchenko, O. Pomorova, Evaluation of Mutual Influences of Software Quality Characteristics Based ISO 25010:2011, in: Proceedings of 2016 International Scientific and Technical Conference. Computer Science and Information Technologies, CSIT'2016, Lviv, 2016, pp. 80-83. doi: 10.1109/STC-CSIT.2016.7589874.

47. T. Hovorushchenko, O. Pomorova, Information Technology of Evaluating the Sufficiency of Information on Quality in the Software Requirements Specifications. CEUR-WS 2104 (2018), pp. 555-570.

48. I. Margarido, J. Faria, R. Vidal, M. Vieira, Classification of Defect Types in Requirements Specifications: Literature Review, Proposal and Assessment, in: Proceedings of 2011 Iberian Conference on Information Systems and Technologies, CISTI'2011, Chaves, 2011, pp. 1-6.

49. Cost of a bug within a software lifecycle. Web-site. URL: http://www.testically.org/2012/02/09/cost-of-a-bug-within-a-software-lifecycle/ (Last accessed: December 7, 2017), pp. 158-161.

50. ISO/IEC 25010:2011. Systems and software engineering. Systems and software Quality Requirements and Evaluation (SQuaRE). System and software quality models. [Introduced 01.03.2011]. Geneva (Switzerland), 2011. 34 p. (International standard).

51. ISO/IEC 25030:2007. Software engineering. Software product Quality Requirements and Evaluation (SQuaRE). Quality requirements. [Introduced 01.06.2007]. Geneva (Switzerland), 2007. 36 p. (International standard).

52. Bourque P., Fairley R. E. Guide to the software engineering body of knowledge (SWEBOK): Version 3.0. IEEE Computer Society, 2014. 335 p.

53. ISO/IEC TR 19759:2015. Software Engineering. Guide to the software engineering body of knowledge (SWEBOK). [Introduced 01.10.2015]. Geneva (Switzerland), 2015. 336 p. (International standard).

54. ISO 9000:2015. Quality management systems. Fundamentals and vocabulary. [Introduced 15.09.2015]. Geneva (Switzerland), 2015. 51 p. (International standard).

55. ISO 9001:2015. Quality management systems. Requirements. [Introduced 15.09.2015]. Geneva (Switzerland), 2015. 29 p. (International standard).

56. Грицюк, Ю.І. Аналіз вимог до програмного забезпечення: Навчальний посібник. Львів: Видавництво Львівської політехніки. – 2018. – 460 с.

57. Грицюк Ю.І. Засіб для визначення якості програмного забезпечення методами метричного аналізу / Ю.І. Грицюк, О.Т. Андрущакевич // Науковий Вісник НЛТУ України. 2018, т. 28, №6. С. 159-171. doi.org/10.15421/40280431.

58. Proceedings of International Conference on Emerging Technologies and Intelligent Systems. ICETIS 2021. Volume 2. 1046 p. ISSN2367-3370. https://doi.org/10.1007/978-3-030-85990-9.

59. Intelligent Manufacturing. Springer Nature Singapore Pte Ltd. 2022. 348 p. ISBN978-981-19-0169-0. https://doi.org/10.1007/978-981-19-0167-6.

60. Proceedings of the 4th International Conference on Big Data Analytics for Cyber-Physical System in Smart City – Volume 2. BDCPS 2022, December 16–17, 2022, Bangkok, Thailand. Springer Nature Singapore Pte Ltd. 2023. №XVI, 733 p. ISBN978-981-99-1157-8Published: 31 March 2023. https://doi.org/10.1007/978-981-99-1157-8.

61. Application of Intelligent Systems in Multi-modal Information Analytics. The 4th International Conference on Multi-modal Information Analytics (ICMMIA 2022), Volume 2. Springer Nature Switzerland AG 2022. №XXV, 1112 p. ISBN978-3-031-05484-6Published: 13 June 2022. https://doi.org/10.1007/978-3-031-05484-6.

62. The Effect of Information Technology on Business and Marketing Intelligence Systems. Springer Cham. №XXIV, 2580 p. ISBN978-3-031-12381-8Published: 09 February 2023. https://doi.org/10.1007/978-3-031-12382-5.

63. Proceedings of the International Conference on Internet of Things, Communication and Intelligent Technology. Springer Singapore. №XV, 794 p. ISSN1876-1100. https://doi.org/10.1007/978-981-99-0416-7.

64. Advances in Intelligent Automation and Soft Computing. Springer Cham. ISBN978-3-030-81006-1Published: 27 July 2021. №XXIX, 1304 p. https://doi.org/10.1007/978-3-030-81007-8.

65. Application of Intelligent Systems in Multi-modal Information Analytics. The 4th International Conference on Multi-modal Information Analytics (ICMMIA 2022), Volume 1. №XXIII, 1058 p. ISBN978-3-031-05237-8Published: 07 May 2022. https://doi.org/10.1007/978-3-031-05237-8.

66. Information and Communication Technology for Intelligent Systems. Proceedings of ICTIS 2020, Volume 1. №XVI, 780 p. ISBN978-981-15-7080-3Published: 23 October 2021. https://doi.org/10.1007/978-981-15-7078-0.

67. Intelligent Systems Design and Applications. 20th International Conference on Intelligent Systems Design and Applications (ISDA 2020) held December 12-15, 2020. №XXII, 1419 p. ISBN978-3-030-71187-0Published: 02 June 2021. https://doi.org/10.1007/978-3-030-71187-0.

68. Proceedings of 2023 Chinese Intelligent Automation Conference. Springer Singapore. №XII, 842 p. ISBN978-981-99-6186-3Published: 23 September 2023. https://doi.org/10.1007/978-981-99-6187-0.

69. Proceedings of Third International Conference on Intelligent Computing, Information and Control Systems. ICICCS 2021. №XX, 1074 p. ISBN978-981-16-7330-6Published: 14 March 2022. https://doi.org/10.1007/978-981-16-7330-6.

70. Intelligent Sustainable Systems. Proceedings of ICISS 2023. №XXX, 868 p. ISBN978-981-99-1726-6Published: 15 June 2023. https://doi.org/10.1007/978-981-99-1726-6.

71. Innovative Computing Vol 1 – Emerging Topics in Artificial Intelligence. Proceedings of IC 2023. №XVI, 1026 p. ISBN978-981-99-2094-5Due: 15 May 2024. https://doi.org/10.1007/978-981-99-2092-1.

72. Intelligent Sustainable Systems. Selected Papers of WorldS4 2022, Volume 2. №XXXI, 783 p. ISBN978-981-19-7663-6Published: 24 January 2023. https://doi.org/10.1007/978-981-19-7663-6.

73. Innovative Technologies in Intelligent Systems and Industrial Applications. CITISIA 2022. №XIX, 1009 p. ISBN978-3-031-29080-0Due: 19 October 2024. https://doi.org/10.1007/978-3-031-29078-7.

74. Google Workspace Learning Centre. [Online]. https://support.google.com/a/users/answer/7212025?hl=en [Accessed 29 November 2023].

ДОДАТОК А

КОПІЇ ОПУБЛІКОВАНИХ НАУКОВИХ СТАТЕЙ

IEEE

Proceedings of the The 12th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)

Volume 1

IDAACS'2023

The crossing point of Intelligent Data Acquisition & Advanced Computing Systems and East & West Scientists

September 7-9, 2023
Dortmund, Germany

ORGANIZED AND SPONSORED BY

Dortmund University of Applied Sciences and Arts, Dortmund, Germany
ruhrvalley Cluster e.V. The DeepTech Innovation Network
EuroPIM – European Partnership for Project and Innovation Management,
Research Institute for Intelligent Computer Systems,
West Ukrainian National University and V.M. Glushkov Institute of Cybernetics, National Academy for Sciences of Ukraine, Ukraine
Faculty of Computer Information Technologies, West Ukrainian National University
IEEE Ukraine Section I&M / CI Joint Societies Chapter,
IEEE Germany Section I&M Society Chapter
MagneticOne, MDPI Sensors, River Publishers, UNITY, Smart Mechatronics, adesso, Yaware

Table of Contents

Volume 1

The 12th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications
7-9 September, 2023, Dortmund, Germany

Intelligent Information Technology for Identification of Remaining Defects in Software Products

Iryna Zasornova [1], Tetiana Hovorushchenko [2], Mykola Fedula [3], Viktoriia Buzyl [4]
[1, 2, 3] Khmelnytskyi National University, 11, Institutska str., Khmelnytskyi, 29016, Ukraine, izasornova@gmail.com, tat_yana@ukr.net, mailfm2000@gmail.com
[4] Ruby Play Company, Elite Business Centre, Trejqa ta Box Box Msida, MSD 1840, Malta, vika.ab53@gmail.com

***Abstract*—An actual task nowadays is to identify remaining defects in software products after their testing. The purpose of this study is to develop an intelligent information technology for identification of remaining defects in software products. This paper's scientific novelty is the developed structure of formation and a detailed scheme of information technology for identification of remaining defects in software products. The intelligent information technology for identification of remaining defects in software products provides the conclusion about the remaining defects' absence or presence (with different degrees of criticality of their consequences on the basis of a natural language report on defects in a software product identified during its main testing).**

***Keywords*—software products testing; remaining defects in software products; intelligent information technology; degree of criticality of the consequences of remaining defects**

I. Introduction & State-of-the-Art

In recent years, software products use has significantly increased in almost all subject areas (medicine, education, finance, trade, management, construction, agriculture, etc.) [1] - [7]. Each software product must comply with a given specification, which guarantees quality and compliance with the specified customer requirements [8] – [10]. At the same time, testing is a necessary process of creating a software product.

Today, there are a lot of testing methods and tools. Most large IT-companies try to use fully automated testing. Researchers and scientists also pay considerable attention to automated testing. This is due to the rapid development of tools for automated testing, improvement of software development methodologies, and testing in particular. Using automated testing, it is possible to achieve more accurate results, reduce the percentage of manual actions (which is especially important given the increasing complexity and size of modern software products, which lead to an increase in the complexity of their testing), and save the project budget [11].

The advantages of known tools for automated testing include [12] – [15]: high speed of test cases execution; absence of human factor influence; minimization of costs and human participation; ability to process large amounts of data.

The disadvantages of known automated testing tools include [12] – [15]: the need for highly qualified personnel and the need to train personnel to use them; high costs for their development and implementation; the need for more comprehensive risk management; unsuitability in case of significant changes in requirements; rapid obsolescence when changing the technological process; inability to identify remaining defects.

Today, start-ups, and small and medium-sized IT-companies often need help with human resources that can be used to automated testing of software products, which leads to the inability to solve the testing tasks [16].

Even if software testing is carried out in full, it only sometimes guarantees the absence of software products' defects [17]. Defects can remain in software products due to: lack of testing funds, incomplete tests, imperfect testing methods, insufficient qualifications of testers and developers, the influence of their inherent disadvantages, etc. [18] – [20]. Such defects are the remaining defects.

A *remaining defect* is software product's defect that remains in software after it has been tested and debugged. The remaining defects in software products are fraught with their failures or malfunctions, which may result in accidents and disasters, reputational losses, information losses, financial losses, and human losses.

The results of the presence of remaining defects in software products are shown in Table 1.

The steady increase in the number of LOC (lines of source code) [21] and the growing number of stories about software defects [22] suggest the potential growth in the amount of remaining software defects.

According to Undo & Cambridge Business School report [23] (Fig. 1), 26% of the total software project time, and USD 61 billion of the project budget are spent on identifying the software product defects (including remaining ones).

979-8-3503-5805-6/23/$31.00 ©2023 IEEE

TABLE I. RESULTS OF REMAINING DEFECTS IN SOFTWARE PRODUCTS

Event	Consequence	Losses
Remaining defects in the computerized voting system Shadow [24]	Inability to hold the 2020 USA Democratic Party vote	Reputational losses
Failures in information systems of British Airways during flight service and online check-in in 2019 [25]	Cancellation of 117 flights at Heathrow, 10 flights at Gatwick	USD 200 000 000
Remaining defects in the timing of computerized system of Boeing CST-100 Starliner spaceship's first launch in 2019 [26]	Failure to reach the required orbit and misalignment with the International Space Station	Financial and reputational losses
Profits loss and inability to collect on loans for Provident Financial in 2018 [27] as a result of remaining defects	The falling share price of Provident Financial (British financial company)	USD 2.2 billion
709 thousand letters with medical data were not delivered to patients or their doctors in the UK in 2018 due to remaining defects [27]	1700 cases with the serious deterioration in patients' health	Reputational losses, human losses
Increase in prison sentences of inmates in the range of 0.5 to 1.5 years in the USA in 2018 [27]	Hundreds of lawsuits filed by inmates	Reputational losses
Remaining software defect in Fiat Chrysler trucks in 2017 caused airbags and seatbelts to disengage [28]	Fatal crashes, recall of 1.25 million sold trucks by Fiat Chrysler	Human, financial, and reputational losses

Figure 1. Time and money spent on defects identification [23]

So, an *actual task* nowadays is to identify remaining defects in software products after their testing, and *our study is aimed* to development of the intelligent information technology for identification of remaining defects in software products.

II. INTELLIGENT INFORMATION TECHNOLOGY FOR IDENTIFICATION OF REMAINING DEFECTS IN SOFTWARE PRODUCTS

Let's classify remaining defects by the degree of criticality of their consequences: minor (the degree of criticality of their consequences is 1) – defects in a software product, in the presence of which the software product is suitable for use without loss of functionality; moderate (the degree of criticality of their consequences is 2) – defects in a software product, in the presence of which the software product is usable, but there is a partial loss of functionality; serious (the degree of criticality of their consequences is 3) – defects in a software product, in the presence of which the software product is unusable due to generating the false results; catastrophic (the degree of criticality of their consequences is 4) – defects of a software product, in the presence of which the software product is unusable due to data distortion and causing system failure.

As it was proved in [29], the accumulation of defects with a certain, lower degree of criticality of their consequences can cause the emergence of remaining defects with subsequent, higher degrees of the criticality of their consequences. In [29], the following general rule was proposed: if the ratio of i-th criticality degree defects' total value to the detected defects' total number (or i-th criticality degree defects' total value) exceeds threshold r_i, then there are remaining defects with the $(i+1)$-th degree of criticality of their consequences. Empirically, in [29], thresholds r_i were established for each type of remaining defects by the degree of criticality of their consequences: $r_1 = 0.75$ (if the ratio of 1st criticality degree defects' total value to the detected defects' total number is equal to or greater than 0.75, there are 2nd criticality degree remaining defects in the software product); $r_2 = 0.5$ (if the ratio of 2nd criticality degree defects' total value to the detected defects' total number is equal to or exceeds 0.5, there are 3rd criticality degree remaining defects in the software product); $r_3 = 2$ (if 3rd criticality degree defects' total value is equal to or greater than 2, there are 4th criticality degree remaining defects in the software product); $r_4 = 1$ (if 4th criticality degree defects' total value is equal to or greater than 1, the software product is unusable and may lead to system failure).

Thus, to identify remaining defects in software products with different degrees of the criticality of their consequences, the defects' information (their types and number) of the software product that were detected during its testing is necessary. Based on the such natural language information, we need to draw a conclusion on the presence or absence of remaining defects with different degrees of the criticality of their consequences. Such a task is difficult to formalize due to the complexity of processing the initial data on existing defects and the need to consider the

mutual influence of detected and remaining defects in the software product, etc. Therefore, to solve this problem, we will use an artificial neural network (Fig. 2).

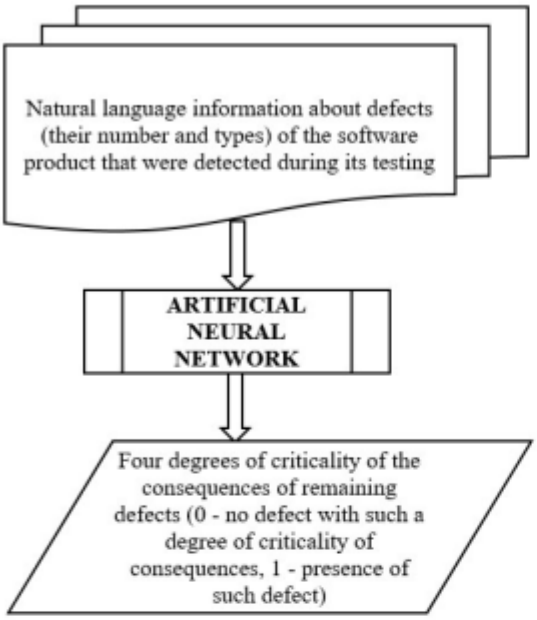


Figure 2. Concept of identification of remaining defects in software products

Thus, the concept underlying intelligent information technology for identification of remaining defects in software products (Fig. 3) is the identification of remaining defects with different degrees of the criticality of their consequences based on information about the defects in the software product identified during its testing.

Intelligent information technology for identification of remaining defects in software products is grounded on the following principles of design and operation: the principle of development (updating the composition and functions of information technology without disrupting its functioning); the principle of compatibility (the possibility of interaction of information technology with other information technologies); the principle of automation of information processing (use of technical tools at all stages of information flow); the principle of efficiency (maximum effect with minimum costs); the principle of stages (the possibility of consistent development of information technology); the principle of systematicity (a single methodological approach that considers the object of research as a whole); the principle of openness (ensuring the truthfulness, veracity, efficiency, regularity, and reliability of information).

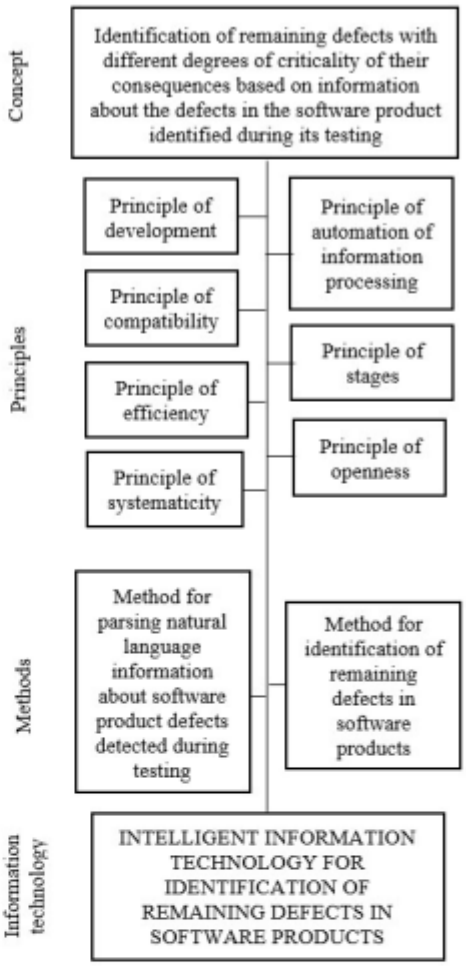


Figure 3. Structure of formation of intelligent information technology for identification of remaining defects in software products

Method for parsing natural language information on software product defects, which are detected during testing, and method for identification of remaining defects in software products were developed by authors in [29].

A detailed scheme of intelligent information technology for identification of remaining defects in software products is represented in Fig. 4.

The intelligent information technology for identification of remaining defects in software products provides a conclusion about the absence or presence of remaining defects with different degrees of the criticality of their consequences on the basis of a natural language report on defects in a software product identified during its main testing.

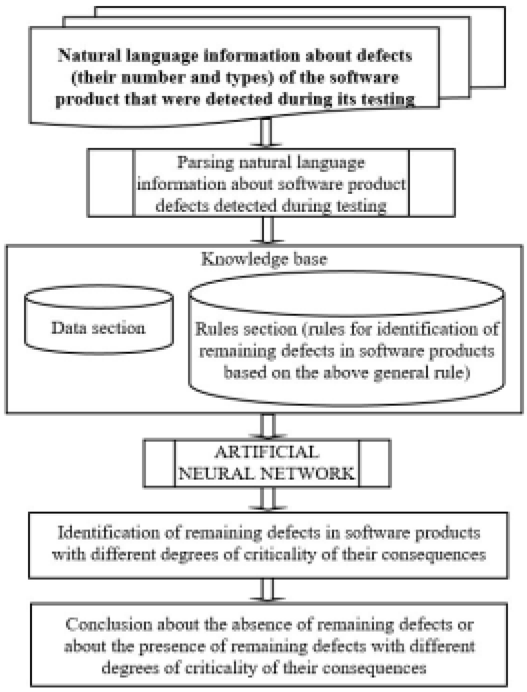


Figure 4. Detailed scheme of intelligent information technology for identification of remaining defects in software products

III. Case Study

In the *first case*, the developed information technology processed natural language information about defects in the software part of the cyberphysical system "Smart Parking", identified during its main testing. Parsing of this natural language information was performed, information about the identified software product's defects was selected, and this information was converted from linguistic representation into a quantitative form with the knowledge base tables, and the input vectors of the ANN were formed. The ANN processed the received input information and provided the following results: {[0;0;1;0] [0;1;0;0] [1;0;0;0] [0;1;0;0] [1;0;0;0] [1;0;0;0] [1;0;0;0] [0;1;0;0] [0;1;0;0] [0;1;0;0] [1;0;0;0] [0;0;1;0] [0;1;0;0] [1;0;0;0] [1;0;0;0] [0;1;0;0] [1;0;0;0] [0;1;0;0] [0;1;0;0] [0;1;0;0]}.

So, taking into account the results provided by ANN, it is evident that two defects with a 3rd degree of criticality of their consequences (serious), ten defects with a 2nd degree of criticality of their consequences (moderate), eight 1st criticality degree defects (minor) were identified. Since the ratio of 1st criticality degree defects' total value to the detected defects' total number is 8/20=0.4, that is, does not exceed 0.75, there are no remaining defects with the 2nd degree of criticality of their consequences in the analyzed software product. Since the ratio of 2nd criticality degree defects' total value to the detected defects' total number is 10/20=0.5, then the analyzed software product has remaining defects with the 3rd degree of criticality of their consequences. Since 3rd criticality degree defects' total value is 2, the analyzed software product has 4th criticality degree remaining defects.

Thus, the proposed intelligent information technology for identification of remaining defects in software products provides a conclusion on the presence of remaining defects with the 3rd and 4th degrees of the criticality of their consequences in the software part of the cyberphysical system "Smart Parking".

In the *second case*, the developed information technology processed natural language information about defects in the software part of the lighting control subsystem of the cyberphysical system "Smart House", identified during its main testing. Parsing of this natural language information was performed, information about the identified software product's defects was selected, and this information was converted from the linguistic representation into quantitative form with knowledge base tables, and the input vectors of the ANN were formed. The ANN processed the received input information and provided the following results: {[1;0;0;0] [1;0;0;0] [1;0;0;0] [1;0;0;0] [1;0;0;0] [1;0;0;0] [1;0;0;0] [1;0;0;0] [1;0;0;0] [0;1;0;0] [1;0;0;0] [1;0;0;0] [0;1;0;0] [1;0;0;0] [1;0;0;0] [0;1;0;0] [1;0;0;0] [0;1;0;0] [0;1;0;0] [0;1;0;0] [1;0;0;0] [0;1;0;0] [1;0;0;0] [1;0;0;0] [0;1;0;0] [1;0;0;0] [0;1;0;0] [0;1;0;0] [0;1;0;0] [1;0;0;0] [1;0;0;0] [0;1;0;0] [1;0;0;0] [1;0;0;0] [1;0;0;0] [1;0;0;0] [1;0;0;0] [1;0;0;0] [1;0;0;0] [1;0;0;0] [1;0;0;0] [1;0;0;0] [1;0;0;0] [1;0;0;0] [1;0;0;0] [1;0;0;0] [1;0;0;0] [1;0;0;0] [1;0;0;0]}.

So, taking into account the results provided by ANN, it is evident that twelve defects with the 2nd degree of criticality of their consequences and thirty-seven defects with the 1st degree of criticality of their consequences were identified. Since the ratio of 1st criticality degree defects' total value to the detected defects' total number is 37/49=0.76, that is, exceed 0.75, there are 2nd criticality degree remaining defects in the analyzed software product. Since the ratio of 2nd criticality degree defects' total value to the detected defects' total number is 12/49=0.24, that is, not exceed 0.5, then the analyzed software product has no remaining defects with the 3rd degree of criticality of their consequences.

So, the proposed intelligent information technology for identification of remaining defects in software products provides a conclusion on the presence of 2nd criticality degree remaining defects in the software part of the lighting control subsystem of the cyberphysical system "Smart Parking".

IV. Conclusions

An actual task nowadays is to identify remaining defects in software products after their testing. This study is aimed to development of the intelligent information technology for identification of remaining defects in software products.

This paper's scientific novelty is the developed structure of formation and a detailed scheme of information technology for identification of remaining defects in software products. The intelligent information technology for identification of remaining defects in software products provides a conclusion on the absence or the presence of remaining defects with different degrees of the criticality of their consequences on the basis of a natural language report on defects in a software product identified during its main testing.

Two cases of using intelligent information technology for identification of remaining defects in software products are considered. For the first case, the proposed intelligent information technology for identification of remaining defects in software products provides a conclusion on the presence of remaining defects with the 3rd and 4th degrees of the criticality of their consequences in the software part of the cyberphysical system "Smart Parking". For the second case, the proposed intelligent information technology for identification of remaining defects in software products provides a conclusion on the presence of remaining defects with the 2nd degree of the criticality of their consequences in the software part of the lighting control subsystem of the cyberphysical system "Smart Parking".

REFERENCES

[1] L. Li, and Y. Tian, "Application of Data Guidance Site Generation Technology in the Cloud Platform Supporting the Construction of Subject Teams in Finance and Economics Applied Universities", *2022 International Conference on Edge Computing and Applications, India*, pp. 19-22, November 2022.

[2] K. Król, and D. Zdonek, "Local Government Website Accessibility—Evidence from Poland" *Administrative sciences*, vol. 10, paper no. 22, 2020.

[3] K. Kirk, and A. S. Abrahams, "Evaluating public charity websites. Stage Model versus Automated Service", *Nonprofit Management & Leadership*, pp. 475-491, 2017.

[4] Y. Akgül, "Web Accessibility Evaluation of Government Websites for People with Disabilities in Turkey", *Journal of Advanced Management Science*, vol. 4, no. 3, pp. 201-210, 2016.

[5] K. Fatima, N. Z. Bawany, and M. Bukhari, "Usability and Accessibility Evaluation of Banking Websites", *2020 International Conference on Advanced Computer Science and Information Systems, Indonesian*, pp. 247-256, November 2020.

[6] M. Dyvak, N. Porplytsya, I. Borivets and M. Shynkaryk, "Improving the computational implementation of the parametric identification method for interval discrete dynamic models," *2017 12th International Scientific and Technical Conference on Computer Sciences and Information Technologies (CSIT), Lviv, Ukraine*, pp. 533-536, 2017.

[7] M. Dyvak, O. Kozak, A. Pukas, "Interval model for identification of laryngeal nerves," *Przegląd Elektrotechniczny*, vol. 86, no. 1, pp. 139-140, 2010.

[8] T. Hovorushchenko, "Methodology of Evaluating the Sufficiency of Information for Software Quality Assessment According to ISO 25010", *Journal of Information and Organizational Sciences*, vol. 42, no.1, pp. 63-85, 2018.

[9] M.M. Şimşek, T. Ergun, and H. Temuçin, "SSL Test Suite: SSL Certificate Test Public Key Infrastructure", *2022 30th Signal Processing and Communications Applications Conference, Turkey*, pp. 1-4, May 2022.

[10] T. Hovorushchenko, O. Pavlova, and D. Medzatyi, "Ontology-Based Intelligent Agent for Determination of Sufficiency of Metric Information in the Software Requirements", *Advances in Intelligent Systems and Computing*, vol. 1020, pp. 447-460, 2020.

[11] M. Rennhard, M. Kushnir, O. Favre, D. Esposito, and V. Zahnd, "Automating the Detection of Access Control Vulnerabilities in Web Applications", *SN Computer Science*, vol. 3, paper no. 376, 2022.

[12] H.-P. Nguyen, T.-N. Luong, and N.-T. Truong, "Generating Test Paths to Detect XSS Vulnerabilities of Web Applications", *2022 9th NAFOSTED Conference on Information and Computer Science, Vietnam*, pp. 287-293, 2022.

[13] M. Felderer, P. Zech, R. Breu, M. Büchler, and A. Pretschner, "Model-based security testing: a taxonomy and systematic classification", *Software Testing, Verification & Reliability*, vol. 26, issue 2, pp. 119-148, 2016.

[14] S. Nair, J. L. de la Vara, M. Sabetzadeh, and L. Briand, "Classification, Structuring, and Assessment of Evidence for Safety - A Systematic Literature Review", *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation, Luxembourg*, pp. 94-103, March 2013.

[15] M. Felderer, B. Agreiter, P. Zech, and R. Breu, "A Classification for Model-Based Security Testing", *The Third International Conference on Advances in System Testing and Validation Lifecycle, Spain*, pp. 109-114, December 2011.

[16] M. Farokhad, J. Otegi-Olaso, L. Pinilla, N. Gandarias, and L. de Lacalle, "Assessing the Success of R&D Projects and Innovation Projects through Project Management Life Cycle", *2019 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, France*, pp. 1104-1110, September 2019.

[17] M. Naidu, "Classification of defects in software using decision tree algorithm", *International Journal of Engineering Science and Technology*, vol. 5, no.06, pp. 11-23, 2013.

[18] R. Natella, S. Winter, D. Cotroneo, and N. Suri, "Analyzing the Effects of Bugs on Software Interfaces", *IEEE Transactions on Software Engineering*, vol. 46, no. 3, pp. 280-301, 2020.

[19] E. Zaitseva, V. Levashenko, J. Rabcan, M. Kvassay, and P. Rusnak, "Reliability Evaluation of Multi-State System Based on Incompletely Specified Data and Structure Function", *2019 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, France*, pp. 741-746, September 2019.

[20] M. Duhan, and P. K. Bhatia, "Software Reusability Estimation based on Dynamic Metrics using Soft Computing Techniques", *International Journal of Computing*, vol. 21 no. 2, pp. 188-194, 2022.

[21] "What is the cost of poor software quality in the U.S.?". [Online]. Available: https://www.synopsys.com/blogs/software-security/poor-software-quality-costs-us. [Accessed 9 March 2023].

[22] "Software Fails Watch". [Online]. Available: https://www.tricentis.com/wp-content/uploads/2019/01/Software-Fails-Watch-5th-edition.pdf. [Accessed 9 March 2023].

[23] "What is the actual cost of software failures?'. [Online]. Available: https://undo.io/the-cost-of-software-failures. [Accessed 9 March 2023].

[24] "How tech firm Shadow sought to revolutionize Democratic campaigns – but stumbled in Iowa". [Online]. Available: https://www.washingtonpost.com/technology/2020/02/04/how-tech-firm-shadow-sought-revolutionize-democratic-campaigns-stumbled-iowa. [Accessed 9 March 2023].

[25] "British Airways passengers stranded after IT failures". [Online]. Available: https://www.bbc.com/news/uk-49261497. [Accessed 9 March 2023].

[26] "Starliner anomaly to prevent ISS docking". [Online]. Available: https://spacenews.com/starliner-anomaly-to-prevent-iss-docking. [Accessed 9 March 2023].

[27] "The 2018 Software Fail Watch Awards". [Online]. Available: https://www.tricentis.com/blog/software-fail-awards-2018. [Accessed 9 March 2023].

[28] "Fiat Chrysler is recalling more than 1.25 million pickup trucks worldwide over a software error that "may be related" to a death and two injuries". [Online]. Available: https://www.bbc.com/news/technology-39898319. [Accessed 9 March 2023].

[29] T. Hovorushchenko, and P. Popov, "Method of Developing the Defect-Free Medical Software by Establishing the Presence of Residual Defects", CEUR-WS, vol. 3038, pp. 11-21, 2021.

ISSN 2710-0766
DOI 10.31891/CSIT

THE INTERNATIONAL SCIENTIFIC JOURNAL

COMPUTER SYSTEMS AND INFORMATION TECHNOLOGIES

No 1-2023

МІЖНАРОДНИЙ НАУКОВИЙ ЖУРНАЛ

КОМП'ЮТЕРНІ СИСТЕМИ ТА ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

2023

INTERNATIONAL SCIENTIFIC JOURNAL *ISSN 2710-0766*
«COMPUTER SYSTEMS AND INFORMATION TECHNOLOGIES»

CONTENTS

INTERNATIONAL SCIENTIFIC JOURNAL *ISSN 2710-0766*
«COMPUTER SYSTEMS AND INFORMATION TECHNOLOGIES»

UDC 004.9: 004.05
https://doi.org/10.31891/csit-2023-1-5

Iryna ZASORNOVA, Tetiana HOVORUSHCHENKO, Oleg VOICHUR
Khmelnytskyi National University

STUDY OF SOFTWARE TESTING TOOLS ACCORDING TO THE TESTING LEVELS

Recently, software has been intensively used in almost all areas of business. Testing is an integral process of the software life cycle, during which it is proved that the software meets the specified requirements and needs of the customer, thereby ensuring the quality of the software. The article analyses the tools for software testing with their generalisation by levels of testing.

The study has shown that there are a number of studies aimed at reviewing and classifying software testing tools. The correct choice of software testing tools is one of the vital elements to ensure the quality of the entire project. However, most studies in the field of testing focus on describing testing methods without directly connecting to the tools that are based on these methods.

A specialist's approach to software testing requires additional information about the testing tools currently available. With the increasing complexity of software products and shorter development cycles, it is clear that manual testing cannot deliver the level of quality required by the market. Choosing the wrong testing tools for a project leads to inadequate quality measurements or tool changes during the project. Both wrong choice and change of testing tools during the development process affect the quality of the software product and, as a result, the success of the project as a whole.

The classifiers discussed in this paper can be used to select software testing tools appropriately. On the one hand, it can be useful for navigating a wide range of testing subjects, reducing the time required for specialists to find the right solution. On the other hand, it can be used as a short introduction to the rapidly developing field of testing and available testing tools for those who are not experts in this field. The classification can be applied to testing various software projects, depending on the type of software and development methodology.

Keywords: software, software testing, manual software testing, automated software testing, levels of software testing.

Ірина ЗАСОРНОВА, Тетяна ГОВОРУЩЕНКО, Олег ВОЙЧУР
Хмельницький національний університет

АНАЛІЗ ІНСТРУМЕНТІВ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ВІДПОВІДНО ДО РІВНІВ ТЕСТУВАННЯ

Останнім часом програмне забезпечення (ПЗ) інтенсивно використовується майже в усіх галузях підприємництва. Тестування є невід'ємним процесом життєвого циклу програмного забезпечення, під час якого доводиться, власне, відповідність ПЗ заданим вимогам і потребам замовника, тим самим забезпечується якість ПЗ. В статті проведено аналіз інструментів для тестування ПЗ з узагальненням їх по рівнях тестування.

Дослідження показало, що існує ряд досліджень, спрямованих на огляд і класифікацію інструментів тестування ПЗ. Коректний вибір інструментів для тестування ПЗ є одним із життєво важливих елементів для забезпечення якості усього проекту. Проте більшість робіт у галузі тестування зосереджені на описі методів тестування без прямого підключення до інструментів, які базуються на цих методах.

Підхід фахівця до тестування ПЗ вимагає додаткової інформації про доступні на даний момент інструменти тестування. Із зростаючою складністю програмних продуктів та короткими циклами розробки стає очевидним, що ручне тестування не може забезпечити рівень якості, необхідний для ринку. Неправильний вибір інструментів тестування для проекту призводить до неадекватних вимірювань якості або заміни інструментів під час проекту. Як неправильний вибір, так і зміна інструментів тестування в процесі розробки впливають на якість програмного продукту і, як наслідок, на успіх проекту в цілому. Класифікатори, які розглянуто у роботі, можна використовувати для відповідного вибору інструментів тестування ПЗ. З одного боку, це може бути корисним для орієнтації в широкому предметному полі тестування, скорочуючи час, необхідний спеціалістам для пошуку вірного рішення. З іншого боку, його можна використати як короткий вступ до галузі тестування, що швидко розвивається, і доступних інструментів тестування для тих, хто не є експертом у цій галузі. Проведена класифікація може бути застосована для тестування різноманітних програмних проектів, залежно від виду ПЗ та методології розробки.

Ключові слова: програмне забезпечення (ПЗ), тестування ПЗ, ручне тестування ПЗ, автоматизоване тестування ПЗ, рівні тестування ПЗ.

Introduction

Recently, software has been intensively used in almost all branches of business [1]. Testing is an integral process of the software life cycle, during which the compliance of the software with the specified requirements and needs of the customer is proven, thereby ensuring the quality of the software [2, 3].

Software testing can be manual or automated. In manual testing, testers perform tests manually without using any means of automation. Manual testing is a low-level and simple type of testing that does not require a lot of additional knowledge. However, before you can automate the testing of any application, you must first run a series of manual tests. Manual testing requires more effort, but without it, we cannot be sure whether automation is possible at all. One of the fundamental principles of testing is that 100% automation is impossible. Therefore, manual testing is a necessity [4].

Automated software testing is part of the testing process at the quality control stage in the software development process; it is a type of testing in which testing is performed using various automation tools and scripts. It uses software tools to execute tests and verify execution results, which helps reduce testing time and simplify the testing process.

Automated testing involves the use of special software to control the execution of tests and compare the expected and actual results of the program. This type of testing helps to automate activities that are often repeated, but which, at the same time, are necessary for maximum test coverage of the task. To compile automated tests, a QA specialist must be able to program. Automated tests are full-fledged programs that are simply designed for testing [4].

There are several main types of automated testing: Code-driven testing automation – testing at the level of software modules, classes and libraries (actually, automatic unit tests); graphical user interface testing automation (Graphical user interface testing) – a special program (testing automation framework) allows you to generate user actions – button presses, mouse clicks, and monitor the program's reaction to these actions – whether it meets the specification; automation of API (Application Programming Interface) testing – the software interface of the program, which is used to test interfaces intended for interaction, for example, with other programs or with the user. Here, again, as a rule, special frameworks are used [4].

The advantages of automated testing include:

– speed of execution of test cases is greater than with manual testing;

– lack of influence of the human factor in the process of execution of test cases;

– cost minimization during repeated execution of test cases (i.e. minimal human participation);

– the ability to perform such test cases that cannot be performed manually;

– the ability to collect, store, analyze, aggregate and present large volumes of data in a form convenient for human perception;

– the ability to perform low-level actions with the application, operating system, transmission channels, etc.

Disadvantages of automated tests include:

– the need for highly qualified personnel;

– high costs for complex automation tools, development and maintenance of test case code;

– automation requires more careful planning and risk management;

– complex selection of automation tools, the need to train staff (or find specialists);

– existing test cases may be unusable and outdated in the case of significant changes in requirements, changes in the technological domain, redesign of interfaces (both user and software), etc.

Currently, much attention is paid to the transition to fully automated software testing, as it provides better results, especially for large and super-large software projects, saves time and budget, allows to reduce the routine manual actions of the tester, and reduces the complexity of software testing [5]. Therefore, there is a rapid development of testing tools and methods. Both manual and automated testing can be used at different levels of testing, as well as be part of other types and types of testing.

Therefore, increasing the use of automated software testing is *an actual task* today.

The purpose of this study is to analyze software testing tools according to the level of testing.

Study of Software Testing Tools According to the Testing Levels

Software testing is usually performed by a team of testers or developers in accordance with set requirements (specifications). Depending on the type of software product, the testing method and appropriate software are chosen.

By level, testing is divided into Unit (first level), Integration (second level), and System (third level) [6]. However, acceptance testing (fourth level) is often added to this list [7]. In order to carry out quality testing, it is necessary to make a reasonable choice of testing tools. Using the classification, it is possible to choose the necessary testing tools.

Software testing tools are mostly designed to support one or more methods of testing a software product and can perform different tasks depending on the level of testing.

The conducted research showed that there is a sufficient number of testing tools designed for unit, integration, system, and acceptance testing.

Unit testing can be considered a basic level of testing with the ability to test several modules at the same time, which makes it possible to automate them. The free and open-source software xUnit is for the first level of testing like *Unit testing*, and depends on the programming language. For example, JUnit is designed for the Java programming language, NUnit for .NET, CppUnit, CUnit for C, C++, etc. [8]. In most cases, unit testing is performed by software product developers using automated tests.

The smallest number of tools was found for *Integration testing*. Testing tools for Unit testing (xUnit) are also used at the next (second) level – during Integration testing. This is related to the method of conducting Integration testing. Also using Rational Rose and Cantata++.

It is known that the behaviour of a software product changes to one degree or another after adding to it a new module that was tested at the first level of testing. Therefore, as part of Integration testing, *Regression testing* is carried out in order to check the functionality of the assembled parts (build) of the software product. Such tests are recommended to be performed using automation tools (Selenium, SilkTest, Rational Functional Tester and QEngine, etc.) since their set is constantly increasing during the development of a software product and manual testing is impractical.

System testing includes a set of the following tests: Recovery, Security, Stress and Performance testing [9]. A large number of tools have been identified for System testing, as this level of testing includes several more testing methods.

Acceptance testing is carried out at the last level before deployment. At this stage, a conclusion is made about the acceptance or rejection of the software product. At the same time, developers can be involved only to correct code errors, if any are found. As the project grows, the number of Acceptance testing will also increase. Therefore, for Acceptance testing, it is recommended to use automated testing tools, for example, FitNesse. FitNesse allows the customer of the software product to enter specially formatted input. Input data is interpreted and tests are created automatically. These tests are then run by the system and the output is returned to the customer. The advantage of this approach is fast feedback. The customer can write tests in the form of HTML tables and add, if necessary, additional text. The tool then analyzes the tables, runs the tests, and provides the results in an HTML document. FitNesse supports Java, but versions for C++, C#, Python, Ruby, Delphi, etc. are currently available.

In addition, it is worth noting that acceptance testing is not sufficiently automated today, as there is a lack of tools for this testing method. Therefore, the use of tools for system testing and acceptance testing is quite limited.

At the same time, it should be taken into account that the software for conducting automated testing is constantly improved, supplemented and expanded. Regarding the future solution, the tools presented in the study can be applied at different levels of testing, providing a practical demonstration of their use. Another possible improvement is to conduct a comparison between testing tools that belong to a similar group to present the advantages and disadvantages of a particular tool [10–11].

Consider the following types of testing. Thus, API testing checks the correctness of interaction between system components, UI testing tests the correctness of the graphical interface, and network protocol level testing checks the correctness of data transmission between computers. All these types of testing are important to ensure the quality of the software and its correct operation.

UI testing, i.e. the user interface, can be performed using the following tools: Abbot Java GUI Test Framework, Business Process Testing (BPT), QF-Test, cPAMIE, Jemmy, Quick Test Professional, Rational Functional Tester, Rational Robot, Selenium, SilkTest, TestComplete, TestPartner, TOSCA, Oracle Functional Testing.

The following tools are used for *API testing*: APL Sanity Autotest, Cantata++, CppTest, CppUnit, CUnit, DTM Data Generator, FitNesse, JProbe, JUnit, NUnit, JVerify, TestNG, VectorCAST/C++.

To *test the network protocol level* – in this case, the tool simulates the client part of the system that interacts with the object through network protocols. Tools: Conkormiq Qtronic, DTM DB Stress, HttpUnit, JMeter, LoadRunner, MessageMagic, NeoLoed, Nikto, OpenSTA, OpenTTCN Tester, Oracle Load Testing, QALoad, Rational App Scan, Rational Performance Tester, SilkPerformer, soapUI, The Grinder, WAPT, Weblnspect.

The listed tools have certain *disadvantages*. For example, although *Selenium* software is one of the most popular tools for automated website testing, it has some drawbacks:

– resistance to change: if the website under test undergoes changes in its structure or code, test scripts written using Selenium may become unusable, requiring the rewriting of tests;

– complexity of configuration: to use Selenium, you need to have basic knowledge of programming and environment configuration;

– limitations in operating systems: Selenium can only work on operating systems that support the browsers it works with;

– slow test execution speed: Selenium can sometimes lag during test execution, which can lead to a longer testing process;

– the need to support scripts: in order for Selenium to be able to work with some elements of the website, such as AJAX, it is necessary to have scripts that can be executed in the browser;

– limitation of liability: Selenium is not able to guarantee responsibility for the completeness of testing, so its use should be supplemented by manual testing and other software quality control methods.

Overall, although Selenium has its flaws, it is a pretty effective tool for automated website testing and performance testing.

Let's consider the advantages and disadvantages of the *Business Process Testing (BPT)* tool. Because of the complexity of testing business processes and the many applications involved, using code-based test automation is problematic. Coded test automation takes time to develop and test. With BPT and multiple scenario testing, the time spent building automated coded testing makes it slow and a significant barrier for organizations. Testing multiple applications requires expertise and knowledge of each application. Coders who develop automation do not have deep applied knowledge. With BPT and multiple applications, this increases testing slowdowns. Advantages of BPT: eliminates the need to create a separate automation system; automated testing becomes structured using business components; reduces the effort required to write and maintain test automation scripts BPT is independent of the detailed test script; high reusability with data-driven components: testers do not need technical knowledge in automation. Disadvantages of BPT: an additional license for the BPT Framework must be purchased for test scripts; The BPT Framework can only be used if you have access to Application Lifecycle Management (ALM).

QF-Test from *Quality First Software* is a cross-platform graphical user interface test automation software tool specializing in Java/Swing, SWT, Eclipse and RCP plug-ins, Java applets, Java Web Start, ULC and cross-browser static and dynamic test automation web applications. A sophisticated recognition mechanism provides extreme serviceability and low maintenance costs, which is the most important factor in software testing automation.

The results of the analysis of tools for software testing with their generalization by testing levels are presented in Table 1.

Table 1

Comparative analysis of tools for automated software testing

№	Tools	Criteria					
		stable to change	easy to set up	supports various OS	test performance speed	script support	URL
				UI testing			
1	Abbot Java GUI Test Framework	stable	difficult	works with all platforms that support Java	low	no	http://abbot.sourceforge.net/
2	Business Process Testing (BPT)	unstable if business processes change regularly	complex in case when many business processes need to be configured	may be limited in the OS on which its components can be used	low	may require support for scripts used to configure tests and automate their execution	http://www.hp.com/
3	QF-Test	stable, but if the PP changes frequently, the tests may require frequent modification	difficult, but if the PP changes frequently, the tests may require frequent modification	supports, but some functions may be limited	low	yes	http://www.qfs.de/en/qftest/index.html
4	cPAMIE	stable if the page does not contain dynamic content	medium difficulty	Windows, Linux	very low	can support JavaScript	http://pamie.sourceforge.net/
5	Jemmy	very stable	medium difficulty, requires some additional components and libraries NetBeans RCP	Windows, Linux, Mac OS X	low	supports based on Java	https://jemmy.dev.java.net/
6	Quick Test Professional	stable, but with frequent requires checking and correction	difficult	Windows	low	scripting support is required	http://www.hp.com/
7	Rational Functional Tester	stable	easy, but you need to have experience in test automation	supports	high, but depends on the volume of test scenarios and the complexity of the application	uses Java scripts	http://www.ibm.com/
8	Rational Robot	stable in the	easy	Windows	high	scripting	http://www.ibm.com/

		absence of significant changes in the user interface				support is required	
9	Selenium	stable	easy	supports	high, but depends on the size of the web page, the number of elements	yes	http://seleniumhq.org/
10	SilkTest	unstable	easy	supports	high, but depends on the complexity of software and the number of tests	yes	http://www.borland.com/
11	TestComplete	very stable	easy	supports	high	yes	http://automatedqa.com/products/testcomplete/
12	TestPartner	stable	easy	Windows	high	yes	http://www.microfocus.com/
13	TOSCA	stable	easy	supports	high	yes	http://www.tricentis/com/
14	Oracle Functional Testing	very stable	easy	supports	high	yes	http://www.oracle.com/
15	Canoo WebTest	stable if the structure of the web pages or web application does not change dramatically	easy	Windows, Linux, Mac OS X	high, but depends on the size of the test suites and the complexity of the web application	yes	http://www.testingtoolsguide.net/tools/canoo-webtest/
16	QEngine	stable	easy	Windows, Linux, Mac OS X	high, but depends on the size of the test sets and the complexity of the software	yes	http://www.manageengine.com/products/qengine/index.html
API testing							
17	APL Sanity Autotest	stable	easy	Windows, Linux, Mac OS X	high	yes	http://ispras.linux-foundation.org/index.php/API_Sanity_Autotest
18	Cantata++	stable if additional functions do not require significant changes to the source code	easy	Windows, Linux, Mac OS X	high	yes	http://www.ipl.com/products/tools/pt400.uk.php
19	CppTest	stable	easy	supports	very high	no	http://cpptest.sourceforge.net/
20	CppUnit	stable	easy	supports	high	no	http://cppunit.sourceforge.net/
21	CUnit	stable	easy	supports	very high	no	http://cunit.sourceforge.net/
22	DTM Data Generator	stable	easy	Windows or with the help of virtual machine	high	no	http://www.sqledit.com/dg/

23	FitNesse	very stable	easy	Windows, Linux, Mac OS X	low	так	http://www.fitnesse.org/
24	JProbe	unstable	easy	supports	high	yes	http://www.quest.com/jprobe/
25	JUnit	stable	easy	Windows, Linux, Mac OS X	high	no	http://www.junit.org/
26	NUnit	stable	easy	Windows	very high	yes	http://www.nunit.org/
27	TestNG	very stable	easy	Windows, Linux, Mac OS X	high	yes	http://testing.org/
28	VectorCAST/ C++	stable	easy	Windows, Linux, Mac OS X	high	yes	http://www.vectorcast.com/software-testing-products/c++-unit-testing.php
29	JVerify	stable	easy	supports	high	no	https://jverify.us/
	Network protocol level testing						
30	Conkormiq Qtronic	stable	easy	Windows, Linux, Mac OS X	high	yes	http://www.conformiq.com/qtronic.php
31	DTM DB Stress	stable	easy	Windows, Linux, Mac OS X	high	yes	http://www.sqledit.com/stress/index.html
32	HttpUnit	stable, if the server code changes, the tests may need to be updated	easy	Windows, Linux, Mac OS X	high	yes	http://httpunit.sourceforge.net/
33	JMeter	very stable	easy	Windows, Linux, Mac OS X	high	yes	http://jakarta.apache.org/jmeter/
34	LoadRunner	stable, but changing existing tests can be quite difficult	easy	Windows	high	yes	https://www.hp.com/
35	MessageMagic	stable, but depends on external libraries	easy	supports	high	yes	http://www.elvior.com/messagemagic/
36	NeoLoed	very stable	easy	supports	very high	yes	http://www.neotys.com/
37	Nikto	stable	easy	Windows, Linux, Mac OS X	very high	no	http://cirt.net/nikto2
38	OpenSTA	stable	difficult	Windows, Linux, Mac OS X	low	yes	http://www.opensta.org/
39	OpenTTCN Tester	stable	easy	Windows, Linux, Mac OS X	high	yes	http://www.openttcn.com/
40	Oracle Load Testing	stable	easy	Windows, Linux	high	yes	http://www.oracle.com/
41	QALoad	stable except for new versions	easy	Windows, Unix	high, but depends on the complexity of the tests	yes	http://www.microfocus.com/

					and the amount of data		
42	Rational App Scan	stable if the web application does not undergo significant changes in structure or design	easy	Windows, Linux	high	yes	http://www.ibm.com/
43	Rational Performance Tester	stable	easy	Windows, Linux, Mac OS X	high	yes	http://www.ibm.com/
44	SilkPerformer	stable if changes are not very significant and the test script should not be completely reworked	easy	supports	high	yes	http://www.borland.com/
45	soapUI	stable	easy	Windows, Linux, Mac OS X	high	yes	http://www.soapui.org/
46	The Grinder	stable	difficult	Windows, Linux, Mac OS X	high for small projects	yes	http://grinder.sourceforge.net/
47	WAPT	stable if the changes are not very significant	easy	Windows, Linux, Mac OS X	high	yes	http://loadtestingtool.com/
48	WebInspect	stable if the changes are not very significant	easy	Windows, Linux, Mac OS X	high, but depends on the complexity of the web application, the number of tests and the amount of data	yes	http://www.hp.com/

Results & Discussion

When performing an analysis of testing tools, the following was found: for UI testing – 16 tools; for API testing – 13 tools; for testing the network protocol level – 19 tools.

In general, there are 9 tools for Unit testing, 4 for Integration testing, 15 for System testing, and 20 for Acceptance testing. The diagram of the distribution of tools depending on the level of testing is presented in Fig. 1.

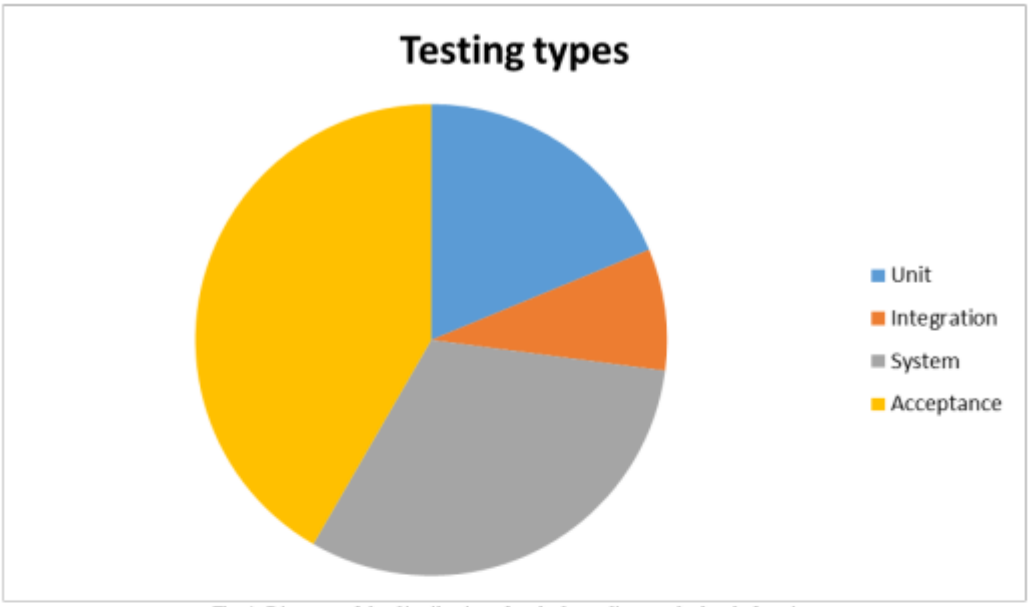


Fig. 1. Diagram of the distribution of tools depending on the level of testing

The conducted research can be used when choosing tools for testing software products.

Conclusions

Recently, software has been intensively used in almost all branches of business. Testing is an integral process of the software life cycle, during which the compliance of the software with the given requirements and needs of the customer is proven, thereby ensuring the quality of the software. The article analyzes software testing tools with their generalization by testing levels.

The study showed that there are a number of studies aimed at reviewing and classifying software testing tools. The correct choice of tools for software testing is one of the vital elements for ensuring the quality of the entire project. However, most of the work in the field of testing focuses on describing test methods without directly connecting to the tools that are based on these methods.

A specialist's approach to software testing requires additional information about currently available testing tools. With the increasing complexity of software products and shorter development cycles, it is becoming apparent that manual testing cannot provide the level of quality required by the market. Choosing the wrong test tools for a project leads to inadequate quality measurements or tool replacement during the project. Both the wrong choice and the change of testing tools during the development process affect the quality of the software product and, as a result, the success of the project as a whole.

The classifiers considered in the work can be used for the appropriate selection of software testing tools. On the one hand, it can be useful for orientation in a wide subject field of testing, reducing the time needed by specialists to find the right solution. On the other hand, it can be used as a brief introduction to the rapidly developing field of testing and available testing tools for those who are not experts in the field. The conducted classification can be used for testing various software projects, depending on the type of software and development methodology.

References

1. L. Li, Y. Tian. Application of Data Guidance Site Generation Technology in the Cloud Platform Supporting the Construction of Subject Teams in Finance and Economics Applied Universities. 2022 International Conference on Edge Computing and Applications: Proceedings (Tamilnadu (India), 2022). Tamilnadu, 2022. Pp. 19-22.
2. T. Hovorushchenko. Methodology of Evaluating the Sufficiency of Information for Software Quality Assessment According to ISO 25010. Journal of Information and Organizational Sciences. 2018. Vol. 42. No.1. Pp. 63-85.
3. T. Hovorushchenko, O. Pavlova, D. Medzatyi. Ontology-Based Intelligent Agent for Determination of Sufficiency of Metric Information in the Software Requirements. Advances in Intelligent Systems and Computing. 2020. Vol. 1020. Pp. 447-460.
4. Manual and automated testing. URL: https://qalight.ua/baza-znaniy/ruchne-ta-avtomatizovane-testuvannya/.
5. M. Rennhard, M. Kushnir, O. Favre, D. Esposito, V. Zahnd. Automating the Detection of Access Control Vulnerabilities in Web Applications. SN Computer Science. 2022. Vol. 3. Paper no. 376.
6. M. Surendra Naidu. Classification of defects in software using decision tree algorithm. International Journal of Engineering Science and Technology. 2013. Vol. 5. Paper no. 06.
7. M. Felderer, P. Zech, R. Breu, M. Büchler, A. Pretschner. Model-based security testing: a taxonomy and systematic classification. Software Testing, Verifiation & Reliability. 2016. Vol. 26. Issue 2. Pp. 119-148.
8. S. Nair, J. L. de la Vara, M. Sabetzadeh, L. Briand. Classification, Structuring, and Assessment of Evidence for Safety - A Systematic Literature Review. 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation: Proceedings (Luxembourg (Luxembourg), 2013). Luxembourg, 2013. Pp. 94-103.

9. M. Felderer, B. Agreiter, P. Zech, R. Breu. A Classification for Model-Based Security Testing. The Third International Conference on Advances in System Testing and Validation Lifecycle: Proceedings (Madrid (Spain), 2011). Madrid, 2011. Pp. 109-114.

10. S. Kumar, S. Mane, S. Mali. A Comparative Study of Machine Learning Algorithms for Software Quality Classification. Journal of King Saud University - Computer and Information Sciences. 2021. Vol. 33. Issue 3. Pp. 308-315.

11. M. Gokhale, S. Shukla. Improving the Software Development Process with SWEBOK and Agile Methodologies. The 4th International Conference on Computing Methodologies and Communication: Proceedings (Erode (India), 2021). Erode, 2021. Pp. 476-484.

Iryna Zasornova **Ірина Засорнова**	Associate Professor of Computer Engineering & Information Systems Department, Khmelnytskyi National University https://orcid.org/0000-0001-6655-5023 e-mail: izasornova@gmail.com	Кандидат технічних наук, доцент, доцент кафедри комп'ютерної інженерії та інформаційних систем, Хмельницький національний університет
Tetiana Hovorushchenko **Тетяна Говорущенко**	DrSc (Engineering), Professor, Head of Computer Engineering & Information Systems Department, Khmelnytskyi National University https://orcid.org/0000-0002-7942-1857 e-mail: govorushchenko@gmail.com	Доктор технічних наук, професор, завідувач кафедри комп'ютерної інженерії та інформаційних систем, Хмельницький національний університет
Oleg Voichur **Олег Войчур**	Lecturer of Computer Engineering & Information Systems Department, Khmelnytskyi National University https://orcid.org/0000-0001-8503-6464 e-mail: o.voichur@gmail.com	Асистент кафедри комп'ютерної інженерії та інформаційних систем, Хмельницький національний університет

ДОДАТОК Б

ХАРАКТЕРИСТИКА МОДЕЛЕЙ ІДЕНТИФІКАЦІЇ ЗАЛИШКОВИХ ДЕФЕКТІВ У ПРОГРАМНОМУ ЗАБЕЗПЕЧЕННІ

Таблиця Б.1 – Характеристика моделей ідентифікації дефектів ПЗ

Модель ідентифікації дефектів ПЗ	Характеристика
1	2
McCall	Значущість: – визначення та ідентифікація дефектів, які впливають на роботу ПЗ в реальних умовах; – оцінка того, наскільки ці дефекти можуть вплинути на користувачів та бізнес-процеси. Коректність: – перевірка ПЗ на наявність дефектів під час розгортання та виходу в експлуатацію; – ідентифікація проблем, які можуть виникнути під час міграції, оновлення або інтеграції програми. Надійність: – оцінка та ідентифікація дефектів, які виявляються після розгортання програми та під час її подальшої експлуатації; – підтримка та виправлення цих дефектів, щоб забезпечити надійну роботу програми
Boehm	Процес розробки: – модель включає методології розробки, такі як модель ЖЦ програми та методи управління ризиками. Ці методології спрямовані на виявлення та запобігання дефектам під час

Продовження таблиці Б.1

1	2
	фаз розробки; Процес тестування: – в моделі надається значна увага тестуванню та валідації ПП. Це допомагає виявляти та документувати дефекти, які виявляються під час тестування, і забезпечує їхнє виправлення до випуску програми. Управління ризиками: – модель підкреслює важливість ідентифікації ризиків і дефектів на ранніх стадіях розробки. За допомогою методів управління ризиками можна ідентифікувати потенційні дефекти та приймати рішення щодо їхнього управління та мінімізації впливу на якість ПЗ. Моніторинг та підтримка: – після випуску програми модель враховує процеси моніторингу та підтримки. Вона допомагає виявляти залишкові дефекти під час експлуатації та вирішувати їх згідно з принципами управління якістю
Dromey	Функціональність: – модель розглядає функціональність програми та ідентифікує дефекти, які можуть впливати на здатність програми виконувати очікувані завдання. Це включає в себе як ідентифікацію дефектів в реалізації функціональних вимог, так і уточнення вимог, які можуть бути недостатньо описаними. Надійність: – враховує надійність програми та ідентифікує дефекти, які можуть призвести до аварій, некоректної роботи або втрати

Продовження таблиці Б.1

1	2
	даних. Легкість використання: – модель оцінює, наскільки легко користувачі можуть взаємодіяти з програмою, і ідентифікує дефекти, що ускладнюють її використання, такі як проблеми з інтерфейсом або незручність взаємодії. Міцність: – цей аспект моделі оцінює здатність програми до підтримки та розвитку. Дефекти, пов'язані із складністю підтримки або розширення програми, також ідентифікуються. Ефективність: – модель враховує ефективність програми у виконанні своїх завдань, а також ідентифікує дефекти, що впливають на продуктивність та використання ресурсів
FURPS	Функціональність: – вказує на функціональні аспекти програми, тобто те, що програма повинна робити. В контексті ідентифікації залишкових дефектів, модель допомагає визначити, чи всі функціональні вимоги виконані належним чином і чи існують недоліки в реалізації функцій. Легкість використання: – стосується зручності та легкості використання програми користувачами. При ідентифікації залишкових дефектів, це важливо для визначення недоліків в інтерфейсі користувача або інших аспектах, які ускладнюють взаємодію з програмою.

Продовження таблиці Б.1

1	2
	Надійність: – вказує на стабільність та безперебійну роботу програми. Ідентифікація залишкових дефектів в цьому контексті полягає у виявленні проблем, які можуть призвести до аварій, помилок або непередбачуваних ситуацій під час використання програми. Продуктивність: – стосується швидкості та ефективності виконання програми. При оцінці залишкових дефектів важливо ідентифікувати проблеми, що впливають на продуктивність та використання ресурсів
ISO 9126	Функціональність: – категорія оцінює функціональність ПП, включаючи його здатність виконувати очікувані завдання. При ідентифікації залишкових дефектів важливо визначити, чи всі функції працюють належним чином та чи відповідають вимогам. Надійність: – цей аспект оцінює стабільність та надійність програми в різних умовах. Дефекти, які можуть призвести до аварій або некоректної роботи, слід ідентифікувати та виправити. Легкість використання: – оцінка легкості використання включає в себе зручність інтерфейсу та способи взаємодії з користувачами. Дефекти в цьому аспекті можуть впливати на сприйняття та задоволення користувачів. Ефективність: – категорія стосується продуктивності та використання

Продовження таблиці Б.1

1	2
	ресурсів під час виконання програми. Дефекти, що впливають на продуктивність, слід виправити. Документація: – категорія стосується наявності та якості документації, яка супроводжує ПП. Дефекти в документації можуть призвести до непорозумінь і помилок в користуванні програмою
ISO 25010	Функціональність: – категорія оцінює функціональність ПП та його здатність виконувати очікувані завдання. Дефекти в цьому аспекті можуть включати невідповідність функціональним вимогам чи помилки в реалізації функцій. Надійність: – оцінює стабільність та безперебійну роботу програми в різних умовах. Дефекти, що можуть призвести до аварій або некоректної роботи, важливо ідентифікувати. Легкість використання: – оцінка легкості використання стосується зручності інтерфейсу та взаємодії з користувачами. Дефекти в цьому аспекті можуть включати проблеми із дизайном інтерфейсу та інші недоліки, які ускладнюють взаємодію з програмою. Ефективність: – оцінює продуктивність програми та використання ресурсів під час виконання завдань. Дефекти, які впливають на продуктивність, важливо ідентифікувати та виправити. Портативність: – стосується здатності програми працювати на різних платформах і в різних середовищах. Дефекти можуть

Кінець таблиці Б.1

1	2
	призвести до некоректної роботи програми. Споживачі ресурсів: – оцінює використання ресурсів, таких як енергія та пам'ять. Дефекти, які призводять до неефективного використання ресурсів, можуть бути ідентифіковані і виправлені. Безпека: – стосується захисту програми та даних від несанкціонованого доступу та атак. Дефекти в цьому аспекті можуть призвести до потенційних порушень безпеки. Суцільність: – оцінює сумісність програми з іншими програмами та середовищами. Дефекти в суцільності можуть впливати на інтеграцію програми з іншими системами.
Bertoa	Документація: – стосується наявності та якості документації. Дефекти можуть призвести до дефектів у користуванні програмою. Функціональність: – точність, придатність, сумісність, відповідність, безпека. Надійність: – зрілість, придатність. Юзабіліті: – навчальність, зрозумілість, працездатність. Ефективність: – поведінка часу, поведінка ресурсів. Ремонтопридатність: – змінюваність, тестоздатність. Портативність: – замінність.

ДОДАТОК В

ЗАСОБИ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ПРОГРАМНИХ ПРОДУКТІВ

Таблиця В.1 – Порівняльний аналіз інструментів для автоматизованого тестування програмних продуктів

№ з/п	Інструменти	Параметри					
		стійкість до змін	легкий у налаштуваннях	підтримує різні ОС	швидкість виконання тестів	підтримка скриптів	URL
1	2	3	4	5	6	7	8
Тестування UI							
1	Abbot Java GUI Test Framework	стійкий	складний	працює з усіма платформами, що підтримують Java	низька	ні	http://abbot.sourceforge.net/
2	Business Process Testing (BPT)	нестійкий, якщо бізнес-процеси регулярно змінюються	складний у випадку, коли потрібно сконфігурувати багато бізнес процесів	може бути обмежений в ОС, на яких можна використовувати його компоненти	низька	може вимагати підтримки скриптів, що використовуються для налаштування тестів та автоматизації	http://www.hp.com/

Продовження таблиці В.1

1	2	3	4	5	6	7	8
3	QF-Test	стійкий, але якщо ПП часто змінюється, то тести можуть вимагати частої модифікації	складний, але його можна налаштувати на потрібний для користувача рівень	підтримує, але деякі функції можуть бути обмеженими	низька	так	http://www.qfs.de/en/qftest/index.html
4	cPAMIE	стійкий, якщо сторінка не містить динамічний контент	має середню складність	Windows, Linux	дуже низька	може підтримувати JavaScript	http://pamie.sourceforge.net/
5	Jemmy	дуже стійкий	середня складність, вимагає деяких допоміжних компонентів та бібліотек NetBeans RCP	Windows, Linux, Mac OS X	низька	підтримує на основі Java	https://jemmy.dev.java.net/
6	Quick Test Professional	стійкий, при частій зміні вимагається перевірка і корекція	складний	Windows	низька	необхідна підтримка скриптів	http://www.hp.com/

Продовження таблиці В.1

1	2	3	4	5	6	7	8
7	Rational Functional Tester	стійкий	легкий, але потрібно мати досвід в автоматизації тестування	підтримує	висока, але залежить від обсягу тестових сценаріїв і складності додатка	використовує скрипти на мові Java	http://www.ibm.com/
8	Rational Robot	стійкий при відсутності значних змін в інтерфейсі користувача	легкий	Windows	висока	необхідна підтримка скриптів	http://www.ibm.com/
9	Selenium	стійкий	легкий	підтримує	висока, але залежить від розміру веб-сторінки, кількості елементів	так	http://seleniumhq.org/
10	SilkTest	нестійкий	легкий	підтримує	висока, але залежить від складності ПЗ та кількості тестів	так	http://www.borland.com/
11	TestComplete	дуже стійкий	легкий	підтримує	висока	так	http://automatedqa.com/products/testcomplete/

Продовження таблиці В.1

1	2	3	4	5	6	7	8
12	TestPartner	стійкий	легкий	Windows	висока	так	http://www.microfocus.com/
13	TOSCA	стійкий	легкий	підтримує	висока	так	http://www.tricentis/com/
14	Oracle Functional Testing	дуже стійкий	легкий	підтримує	висока	так	http://www.oracle.com/
15	Canoo WebTest	стійкий, якщо структура веб-сторінок чи веб-застосунку різко не змінюється	легкий	Windows, Linux, Mac OS X	висока, але залежить від розміру тестових наборів і складності веб-застосунку	так	http://www.testingtoolsguide.net/tools/canoo-webtest/
16	QEngine	стійкий	легкий	Windows, Linux, Mac OS X	висока, але залежить від розміру тестових наборів і складності ПЗ	так	http://www.manageengine.com/products/qengine/index.html
Тестування API							
17	APL Sanity Autotest	стійкий	легкий	Windows, Linux, Mac OS X	висока	так	http://ispras.linux-foundaton.org/index.php/API_Sanity_Autotest
18	Cantata++	стійкий, додаткові функції не	легкий	Windows, Linux, Mac OS X	висока	так	http://www.ipl.com/products/tools/pt400.uk.php

Продовження таблиці В.1

1	2	3	4	5	6	7	8
		вимагають внесення значних змін у вихідний код системи					
19	CppTest	стійкий	легкий	підтримує	дуже висока	ні	http://cpptest.sourceforge.net/
20	CppUnit	стійкий	легкий	підтримує	висока	ні	http://cppunit.sourceforge.net/
21	CUnit	стійкий	легкий	підтримує	дуже висока	ні	http://cunit.sourceforge.net/
22	DTM Data Generator	стійкий	легкий	Windows або з використанням віртуальної машини	висока	ні	http://www.sqledit.com/dg/
23	FitNesse	дуже стійкий	легкий	Windows, Linux, Mac OS X	низька	так	http://www.fitnesse.org/
24	JProbe	нестійкий	легкий	підтримує	висока	так	http://www.quest.com/jprobe/
25	JUnit	стійкий	легкий	Windows, Linux, Mac OS X	висока	ні	http://www.junit.org/
26	NUnit	стійкий	легкий	Windows	дуже висока	так	http://www.nunit.org/
27	TestNG	дуже стійкий	легкий	Windows, Linux, Mac OS X	висока	так	http://testing.org/

Продовження таблиці В.1

1	2	3	4	5	6	7	8
28	Vector CAST /C++	стійкий	легкий	Windows, Linux, Mac OS X	висока	так	http://www.vectorcast.com/software-testing-products/c++-unit-testing.php
29	JVerify	стійкий	легкий	підтримує	висока	ні	https://jverify.us/
Тестування рівня мережевого протоколу							
30	Conkormiq Qtronic	стійкий	легкий	Windows, Linux, Mac OS X	висока	так	http://www.conformiq.com/qtronic.php
31	DTM DB Stress	стійкий	легкий	Windows, Linux, Mac OS X	висока	так	http://www.sqledit.com/stress/index.html
32	HttpUnit	стійкий, якщо серверний код змінюється, то тести можуть вимагати оновлення	легкий	Windows, Linux, Mac OS X	висока	так	http://httpunit.sourceforge.net/
33	JMeter	дуже стійкий	легкий	Windows, Linux, Mac OS X	висока	так	http://jakarta.apache.org/jmeter/
34	LoadRunner	стійкий, але зміна існуючих тестів може бути досить складною	легкий	Windows	висока	так	https://www.hp.com/
35	MessageMagic	стійкий, але залежить	легкий	підтримує	висока	так	

Продовження таблиці В.1

1	2	3	4	5	6	7	8
		від зовнішніх бібліотек					htttp://www.elvior.com/messagemagic/
36	NeoLoad	дуже стійкий	легкий	підтримує	дуже висока	так	http://www.neotys.com/
37	Nikto	стійкий	легкий	Windows, Linux, Mac OS X	дуже висока	ні	http://cirt.net/nikto2
38	OpenSTA	стійкий	складний	Windows, Linux, Mac OS X	низька	так	http://www.opensta.org/
39	OpenTTCN Tester	стійкий	легкий	Windows, Linux, Mac OS X	висока	так	http://www.opentten.com/
40	Oracle Load Testing	стійкий	легкий	Windows, Linux	висока	так	http://www.oracle.com/
41	QALoad	стійкий, окрім нових версій	легкий	Windows, Unix	висока, але залежить від складності тестів та обсягу даних	так	http://www.microfocus.com/
42	Rational App Scan	стійкий, якщо веб-застосунок не зазнає значних змін у структурі або дизайні	легкий	Windows, Linux	висока	так	http://www.ibm.com/
43	Rational	стійкий	легкий	Windows, Linux,	висока	так	http://www.ibm.com/

Кінець таблиці В.1

1	2	3	4	5	6	7	8
	Performance Tester			Mac OS X			
44	SilkPerformer	стійкий, якщо зміни не дуже значні і тестовий сценарій повинен бути не повністю перероблений	легкий	підтримує	висока	так	http://www.borland.com/
45	soapUI	стійкий	легкий	Windows, Linux, Mac OS X	висока	так	http://www.soapui.org/
46	The Grinder	стійкий	складний	Windows, Linux, Mac OS X	висока для малих проєктів	так	http://grinder.sourceforge.net/
47	WAPT	стійкий, якщо зміни не дуже значні	легкий	Windows, Linux, Mac OS X	висока	так	http://loadtestingtool.com/
48	Webln spect	стійкий, якщо зміни не дуже значні	легкий	Windows, Linux, Mac OS X	висока, але залежить від складності веб-застосунка, кількості тестів та обсягу даних	так	http://www.hp.com/

ДОДАТОК Г

ПЕРЕЛІК МОВНИХ ФОРМУЛЮВАНЬ ВИСНОВКІВ ПРО НАЯВНІСТЬ ЗАЛИШКОВИХ ДЕФЕКТІВ У ПРОГРАМНОМУ ЗАБЕЗПЕЧЕННІ

Таблиця Г.1 – Перелік мовних формулювань висновків про наявність залишкових дефектів у програмному забезпеченні

Код правила	Правила формування висновку
0	відсутність залишкових дефектів
1	непридатність ПЗ (можлива відмова системи)
2	наявність залишкових дефектів 4-го рівня критичності (катастрофічних дефектів)
3	непридатність ПЗ (можлива відмова системи) та наявність залишкових дефектів 4-го рівня критичності (катастрофічних дефектів)
4	наявність залишкових дефектів 3-го рівня критичності (серйозних дефектів)
5	непридатність ПЗ (можлива відмова системи) та наявність залишкових дефектів 3-го рівня критичності (серйозних дефектів)
6	наявність залишкових дефектів 3-го та 4-го рівня критичності
7	непридатність ПЗ (можлива відмова системи) та наявність залишкових дефектів 3-го та 4-го рівня критичності
8	наявність залишкових дефектів 2-го рівня критичності (помірних дефектів)
9	непридатність ПЗ (можлива відмова системи) та наявність залишкових дефектів 2-го рівня критичності (помірних дефектів)
10	наявність залишкових дефектів 2-го та 4-го рівня критичності

Кінець таблиці Г.1

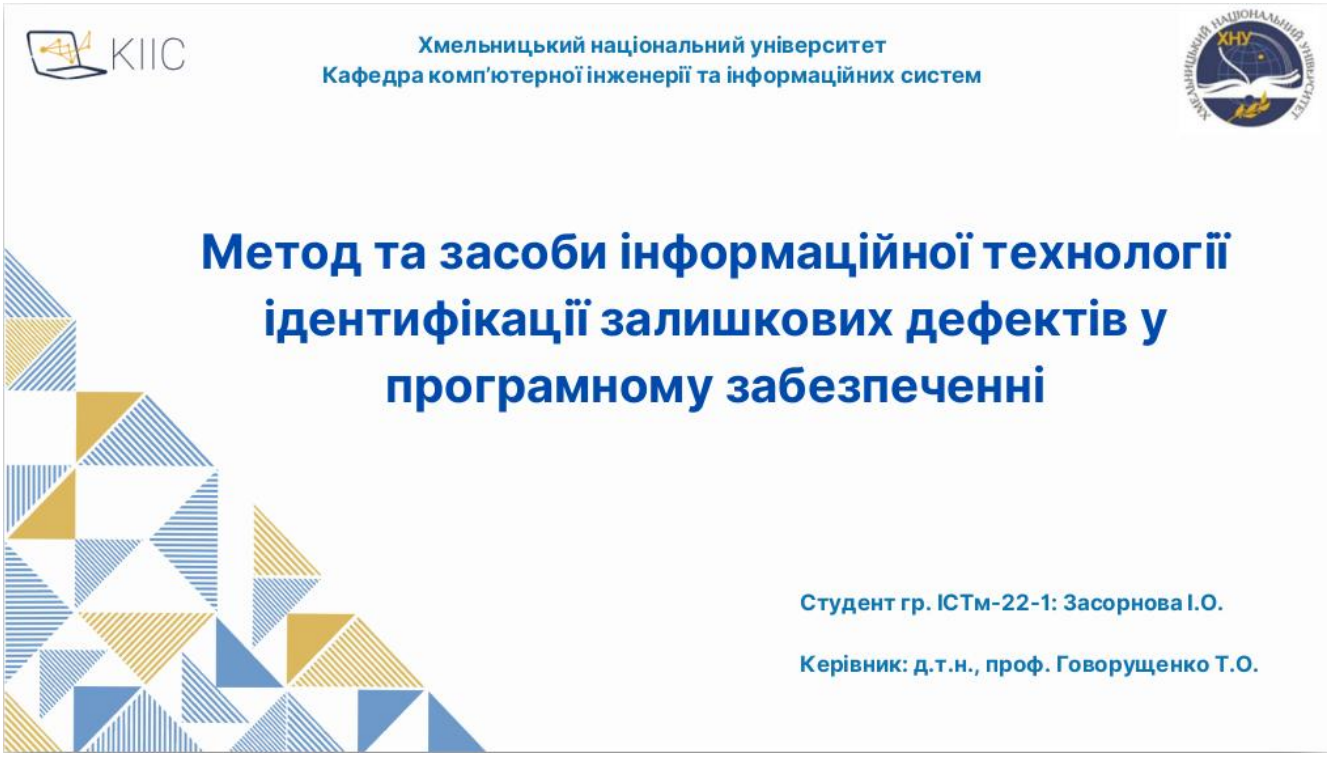
11	непридатність ПЗ (можлива відмова системи) та наявність залишкових дефектів 2-го та 4-го рівня критичності
12	наявність залишкових дефектів 2-го та 3-го рівня критичності
13	непридатність ПЗ (можлива відмова системи) та наявність залишкових дефектів 2-го та 3-го рівня критичності
14	наявність залишкових дефектів 2-го, 3-го та 4-го рівня критичності
15	непридатність ПЗ (можлива відмова системи) та наявність залишкових дефектів 2-го, 3-го та 4-го рівня критичності

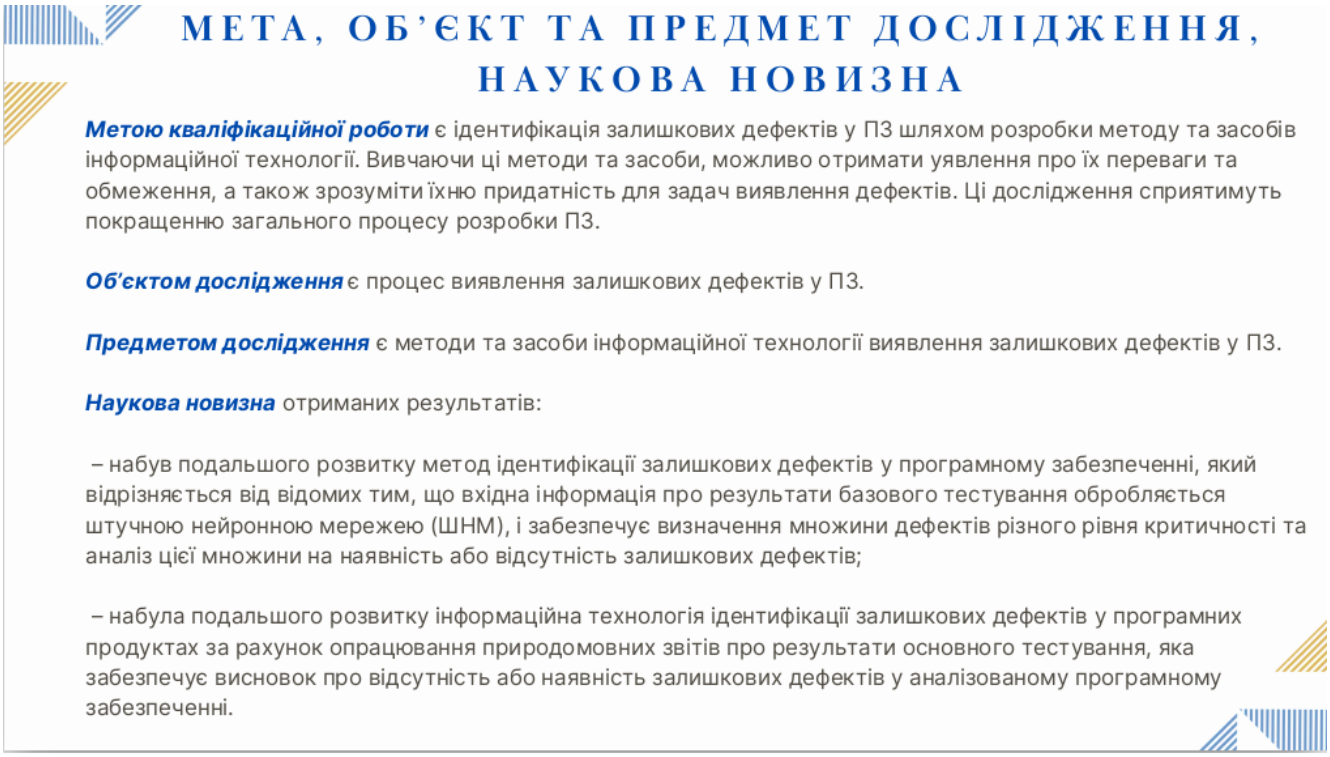
Таблиця Г.2 – Кодування правил формулювань висновків про наявність залишкових дефектів у програмному забезпеченні

Код правила	Рівень критичності дефектів			
	незначні	помірні	серйозні	катастрофічні
	1	2	3	4
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

ДОДАТОК Д

ПРЕЗЕНТАЦІЯ ДО ЗАХИСТУ КВАЛІФІКАЦІЙНОЇ РОБОТИ





ПРАКТИЧНА ЗНАЧУЩІСТЬ ОТРИМАНИХ РЕЗУЛЬТАТІВ, ПУБЛІКАЦІЇ

Практична значущість отриманих результатів полягає у проєктуванні та реалізації інтелектуальної інформаційної технології ідентифікації залишкових дефектів у програмних продуктах, яка забезпечує висновок про відсутність залишкових дефектів або про наявність залишкових дефектів з різними ступенями критичності їх наслідків на основі природомовного звіту про дефекти програмного продукту, виявлені під час його основного тестування.

Публікації. За темою кваліфікаційної роботи опубліковано одну статтю у матеріалах конференції, що індексуються в наукометричній базі Scopus, і одну статтю у фаховому виданні України:

- I. Zasornova, T. Hovorushchenko, M. Fedula, V. Buzyl. Intelligent Information Technology for Identification of Remaining Defects in Software Products // Proceedings of the 2023 IEEE 12-th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS-2023, Dortmund, 7-9 September 2023), vol. 1, pp. 33-37;
- I. Zasornova, T. Hovorushchenko, O. Voichur. Study of Software Testing Tools According to the Testing Levels. Computer Systems & Information Technologies. 2023. №1. Pp. 38-46.

Участь у конференціях. Взято участь у The 12th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS). (September 7–9, 2023, Dortmund, Germany).

АКТУАЛЬНІСТЬ РОБОТИ, ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

В процесі розробки ПЗ залишкові дефекти є суттєвою проблемою, яка може мати значні наслідки для якості та безпеки ПЗ. Залишкові дефекти – це дефекти, які залишаються невиявленими або невирішеними після його тестування та налагодження. Залишкові дефекти в ПП загрожують їм відмовами або неправильною роботою. Це може призвести до часткової (аварійна зупинка) або повної зупинки (катастрофічна зупинка) роботи ПЗ, репутаційних втрат, інформаційних втрат, фінансових втрат і навіть людських жертв. Ці дефекти становлять загрозу для функціональності, продуктивності та надійності ПЗ. Тому, аналіз моделей та засобів виявлення залишкових дефектів має вирішальне значення для підвищення якості ПЗ та забезпечення його надійності.

Отже, ***актуальність роботи*** полягає у виявленні дефектів, що залишилися в ПЗ після їх тестування, відповідно дослідження спрямоване на розробку інформаційної технології для виявлення дефектів, що залишилися в ПЗ.

Для розробки методу і засобів інформаційної технології ідентифікації залишкових дефектів у ПЗ у кваліфікаційній роботі необхідно вирішити наступні ***взаємопов'язані задачі дослідження***:

– проаналізувати існуючі методи та засоби ідентифікації залишкових дефектів у ПЗ;
– визначити напрямки поліпшення методів та засобів ідентифікації залишкових дефектів у ПЗ;
– розробити алгоритм ідентифікації залишкових дефектів у ПЗ;
– розробити метод ідентифікації залишкових дефектів у ПЗ;
– розробити вимоги до інформаційної технології ідентифікації залишкових дефектів у ПЗ;
– розробити структуру інформаційної технології ідентифікації залишкових дефектів у ПЗ;
– представити результати функціонування інформаційної технології ідентифікації залишкових дефектів у ПЗ

КОНЦЕПЦІЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ІДЕНТИФІКАЦІЇ ЗАЛИШКОВИХ ДЕФЕКТІВ У ПЗ

Рисунок 1 – Концепція ідентифікації залишкових дефектів у ПП

Для розробки концепції інформаційної технології ідентифікації залишкових дефектів у ПЗ необхідно виконати класифікацію залишкових дефектів за ступенем критичності їх наслідків. Усі існуючі дефекти за ступенем критичності їх наслідків можна розділити на: незначні, помірні, серйозні, катастрофічні.

Розглянемо більш детально кожну ступінь критичності дефектів:

– незначні (ступінь критичності їх наслідків становить 1) – дефекти ПП, за наявності яких ПП є придатним для використання без втрати функційності;

– помірні (ступінь критичності їх наслідків становить 2) – дефекти ПП за наявності яких ПП є придатним для використання, проте з частковою втратою функційності;

– серйозні (ступінь критичності їх наслідків становить 3) – дефекти ПП, за наявності яких ПП непридатний для використання через генерування хибних результатів;

– катастрофічні (ступінь критичності їх наслідків становить 4) – дефекти ПП, за наявності яких ПП непридатний для використання через спотворення даних та відмови роботи ПЗ.

ПРАВИЛА ФОРМУВАННЯ ВИСНОВКУ ПРО НАЯВНІСТЬ ЗАЛИШКОВИХ ДЕФЕКТІВ

Загальне правило. Якщо відношення сумарного значення дефектів *i*-го рівня критичності до загальної кількості виявлених під час основного тестування дефектів (або сумарне значення дефектів *i*-го рівня критичності) перевищує поріг *ri*, то приймається висновок про наявність залишкових дефектів (*i*+1)-го (наступного) рівня критичності.

Із загального правила витікають наступні конкретні ***правила формування висновку про наявність залишкових дефектів***:

1) Якщо відношення сумарного значення дефектів 1-го рівня критичності (незначних дефектів) до загальної кількості виявлених під час основного тестування дефектів дорівнює або перевищує поріг *r*1, то приймається висновок про наявність залишкових дефектів 2-го рівня критичності (помірних дефектів);

2) Якщо відношення сумарного значення дефектів 2-го рівня критичності (помірних дефектів) до загальної кількості виявлених під час основного тестування дефектів дорівнює або перевищує поріг *r*2, то приймається висновок про наявність залишкових дефектів 3-го рівня критичності (серйозних дефектів);

3) Якщо сумарне значення дефектів 3-го рівня критичності (серйозних дефектів) перевищує поріг *r*3, то приймається висновок про наявність залишкових дефектів 4-го рівня критичності (катастрофічних дефектів);

4) Якщо сумарне значення дефектів 4-го рівня критичності (катастрофічних дефектів) перевищує поріг *r*4, то приймається висновок про непридатність ПЗ і можливу відмову системи.

МОДЕЛЬ І МЕТОД ІДЕНТИФІКАЦІЇ ЗАЛИШКОВИХ ДЕФЕКТІВ У ПРОГРАМНОМУ ЗАБЕЗПЕЧЕННІ

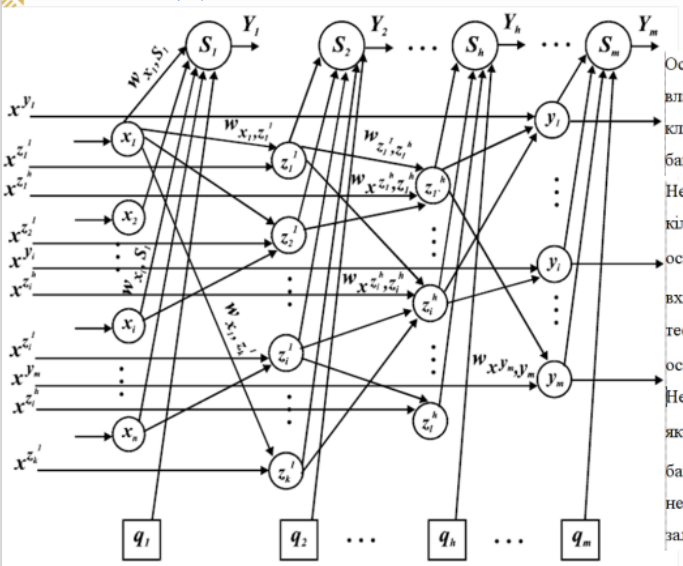


Рисунок 1 – Структура ШНМ для прогнозування залишкових дефектів у ПЗ та визначення їх рівня критичності

Оскільки задачі прогнозування залишкових дефектів у ПЗ не притаманні властивості числового ряду, відсутні обернені зв'язки та немає потреби в класифікації та кластеризації даних, то для розв'язання такої задачі оберемо багатошаровий персептрон.

Нейрони вхідного (рецепторного) шару визначає природомовна інформація про кількість і типи дефектів, виявлених під час основного тестування, тобто звіт основного тестування – множина $X=\{x_i\}, i=(\overline{1,\ z_k})$, де x_i – i-й нейрон вхідного (рецепторного) шару (інформація з i-го рядка звіту основного тестування), кількість вхідних нейронів z_k – кількість рядків аналізованого звіту основного тестування.

Нейрони вихідного (ефекторного) шару визначають множину характеристик якості $Y=\{y_l\}, l=(\overline{1,\ 4})$, де y_l – l-й функціонал ефекторного шару багатошарового персептрону (l-ий рівень критичності), кількість вихідних нейронів наразі становить 4 (оскільки наразі розглядаються 4 рівні критичності залишкових дефектів).

МОДЕЛЬ І МЕТОД ІДЕНТИФІКАЦІЇ ЗАЛИШКОВИХ ДЕФЕКТІВ У ПРОГРАМНОМУ ЗАБЕЗПЕЧЕННІ (2)

Нейрони першого прихованого (апроксимуючого) шару прямонапрямленого персептрона складають множину $\overline{Z^1}=\{z_j^1\},\ j=(\overline{1,k})$, де z_j^1 – j-й нейрон першого прихованого (апроксимуючого) шару, k – кількість нейронів першого прихованого (апроксимуючого) шару (достатньою є кількість нейронів першого прихованого шару $k=(z_k+1)/2$).

Нейрони другого прихованого (коригуючого) шару прямонапрямленого персептрона визначено множиною $\overline{Z^2}=\{z_g^2\},\ g=(\overline{1,k_2})$, де z_g^2 – g-й нейрон другого прихованого (коригуючого) шару, k_2 – кількість нейронів другого прихованого (коригуючого) шару (достатньою є кількість нейронів першого прихованого шару $k_2=(z_k+1)/3$).

Вектор порогових величин зміщень множини нейронних елементів визначено як: $\overline{Q}=\{q_l\},\ l=(\overline{1,\ 4})$, де q_l – зміщення l-го нейронного елемента, $l=4$ – кількість зміщень нейронних елементів (кількість рівнів критичності).

Ваги зв'язків відображено ваговими матрицями: $\overline{W_{X,Z^1}}=\left\langle w_{x_i,z_j^1}\right\rangle,\ i=(\overline{1,\ z_k}), j=(\overline{1,\ k})$, де w_{x_i,z_j^1} – ваговий коефіцієнт зв'язку між x_i-м входом та z_j-м нейроном першого прихованого шару; $\overline{W_{Z^1,Z^2}}=\left\langle w_{z_j^1,z_g^2}\right\rangle,\ j=(\overline{1,k}), g=(\overline{1,k_2})$, де $w_{z_j^1,z_g^2}$ – ваговий коефіцієнт зв'язку між z_j-м нейроном апроксимуючого (першого) прихованого шару та z_g-м нейроном коригуючого (другого) прихованого шару; $\overline{W_{Z^2,Y}}=\left\langle w_{z_g^2,y_l}\right\rangle,\ l=(\overline{1,\ 4})$, де $w_{z_g^2,y_l}$ – ваговий коефіцієнт зв'язку між z_g-м нейроном коригуючого (другого) прихованого шару та l-м нейроном вихідного шару.

МОДЕЛЬ І МЕТОД ІДЕНТИФІКАЦІЇ ЗАЛИШКОВИХ ДЕФЕКТІВ У ПРОГРАМНОМУ ЗАБЕЗПЕЧЕННІ (3)

Формула для визначення i-го функціоналу ефекторного шару ШНМ y_l має вигляд:

$$y_l = f_1\left(f_2(Z^1)\cdot\left(\sum_{j=1}^{k}\sum_{g=1}^{k_2}(z_j^1\cdot w_{z_j^1 z_g^2})\right)+\sum_{i=1}^{Z_k}(x_i\cdot w_{x_i z_j^1})\right)-w_{z_g^2 y_l}, \quad (3.1)$$

де f_1 – активаційна функція нейронів ефекторного шару ШНМ (лінійна функція, результати якої лежать в інтервалі [0; 1]), f_2 – активаційна функція нейронів прихованих шарів (гіперболічний тангенс).

Для опису текстового формування висновку про встановлення наявності залишкових дефектів введемо $f_g \in F$ ($F=\{ f_g \mid g=0..15\}$), де f_g – текстове формулювання висновку про встановлення наявності або відсутності залишкових дефектів, де g – кількість текстових формулювань висновків, що змінюється від 0 до g.

Загальна кількість текстових формувань висновку не може перевищувати шістнадцять, оскільки кількість параметрів, що періряють дорівнює чотирьом ($4^2=16$). Проте, оскільки початкове $g=0$, кінцеве значення індексу $g=15$. Індекс g призначено для вибору з масиву текстових формулювань потрібного висновку.

МОДЕЛЬ І МЕТОД ІДЕНТИФІКАЦІЇ ЗАЛИШКОВИХ ДЕФЕКТІВ У ПРОГРАМНОМУ ЗАБЕЗПЕЧЕННІ (4)

Метод ідентифікації залишкових дефектів у програмному забезпеченні складається з наступних кроків:

1) на основі результатів роботи ШНМ відбувається формування множини $D =\{ di \mid i=1..4\}$, де di – сумарні значення дефектів кожного з рівнів критичності;

2) з використанням множини D відбувається формування множини $DN =\{ dni \mid i=1..4\}$. Перші два члени множини ($dn1$, $dn2$) є відношення $dni = dni / n$, де n – загальна кількість виявлених під час базового тестування дефектів ПЗ. Наступні два члени множини ($dn3$, $dn4$) є сумарні значення дефектів третього і четвертого рівнів критичності, відповідно $dn3 = d3$, а $dn4 = d4$;

3) з використанням множини R (R – множина порогів допустимої кількості дефектів), відбувається порівняння між собою значень елементів множин DN і R (умова порівняння – $dni < ri$) та формування множини $K =\{ ki \mid i=1..4\}$, призначеної для збереження результатів порівняння та формування чотирьохбітного двійкового числа;

4) отримання індексу g для текстової множини $F =\{ fg \mid g=0..15\}$ шляхом перетворення чотирьохбітного двійкового числа в десяткове;

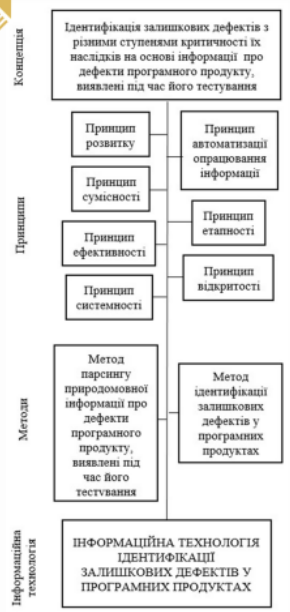
5) вибір з текстової множини F необхідного висновку про наявність або відсутність залишкових дефектів у ПЗ, виведення висновку користувачу та запис у файлі.

МОДЕЛЬ І МЕТОД ІДЕНТИФІКАЦІЇ ЗАЛИШКОВИХ ДЕФЕКТІВ У ПРОГРАМНОМУ ЗАБЕЗПЕЧЕННІ(5)

Отже, вдосконалено нейромережну модель ідентифікації залишкових дефектів у програмному забезпеченні на основі звіту основного тестування, яка відрізняється від відомих тим, що дає можливість враховувати важливість кожного рядка звіту основного тестування, а також взаємний вплив атрибутів в межах дефекту кожного рівня критичності. Вихідні функціонали ШНМ, що відповідають значенням рівнів критичності, дають можливість оцінити сумарний вплив відшуканих під час основного тестування помилок на наявність залишкових дефектів у програмному забезпеченні.

Розроблено метод ідентифікації залишкових дефектів у програмному забезпеченні, суть якого полягає у виявленні множини дефектів різних рівнів критичності та аналізу цієї множини на предмет наявності або відсутності залишкових дефектів.
Метод відрізняється від відомих тим, що вхідна інформація про результати основного тестування опрацьовується штучною нейронною мережею.

СТРУКТУРА ФОРМУВАННЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ІДЕНТИФІКАЦІЇ ЗАЛИШКОВИХ ДЕФЕКТІВ У ПП



Інтелектуальна інформаційна технологія ідентифікації залишкових дефектів у ПП базується на таких принципах проєктування та функціонування:

– принцип розвитку (оновлення складу і функцій інформаційної технології без порушення її функціонування);
– принцип сумісності (можливість взаємодії інформаційної технології з іншими інформаційними технологіями);
– принцип автоматизації опрацювання інформації (використання технічних інструментів на всіх стадіях проходження інформації);
– принцип ефективності (максимальний ефект при мінімальних витратах);
– принцип етапності (можливість послідовного розвитку інформаційної технології);
– принцип системності (єдиний методологічний підхід, який розглядає об'єкт дослідження як єдине ціле);
– принцип відкритості (забезпечення правдивості, достовірності, оперативності, регулярності та надійності інформації).

Рисунок 1 – Структура формування інформаційної технології ідентифікації залишкових дефектів у ПП

СТРУКТУРА ФОРМУВАННЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ІДЕНТИФІКАЦІЇ ЗАЛИШКОВИХ ДЕФЕКТІВ У ПП(2)

Використаємо метод парсингу природомовної інформації про дефекти програмних продуктів, виявлені під час його тестування, та метод ідентифікації залишкових дефектів у програмних продуктах.

Рисунок 1 – Деталізована схема інтелектуальної інформаційної технології ідентифікації залишкових дефектів у ПП

СТРУКТУРА СИСТЕМИ РОЗРОБКИ БЕЗДЕФЕКТНОГО ПЗ ШЛЯХОМ ВСТАНОВЛЕННЯ НАЯВНОСТІ ЗАЛИШКОВИХ ДЕФЕКТІВ

Інтелектуальна інформаційна технологія ідентифікації залишкових дефектів у ПЗ забезпечує висновок про відсутність залишкових дефектів або про наявність залишкових дефектів з різними ступенями критичності їх наслідків на основі природомовного звіту про дефекти ПЗ, виявлені під час його базового тестування.

Як інструмент інформаційної технології ідентифікації залишкових дефектів у ПП запропонуємо систему розробки бездефектного ПЗ шляхом встановлення наявності залишкових дефектів.

Рисунок 1 – Структура системи розробки бездефектного ПЗ шляхом встановлення наявності залишкових дефектів

РЕЗУЛЬТАТИ ФУНКЦІОНУВАННЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ІДЕНТИФІКАЦІЇ ЗАЛИШКОВИХ ДЕФЕКТІВ У ПЗ

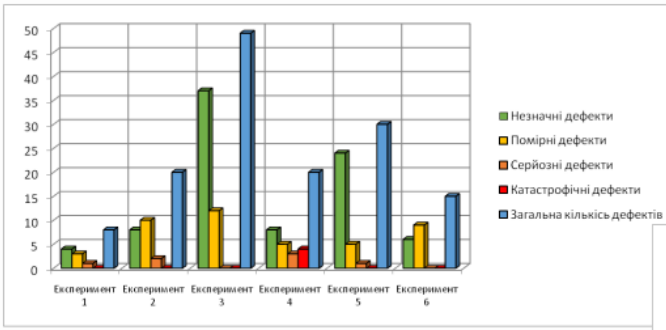


Рисунок 1 – Діаграма виявлених дефектів ПЗ різних рівнів критичності у результаті 6-х проведених експериментів

Апробовано розроблену інформаційну технологію ідентифікації залишкових дефектів у ПП з проведенням десяти експериментів, а саме: шість експериментів ПЗ («Ресторан», «Розумна парковка», «Розумний будинок», «Розумна мийка», «Флорист», "Фільм») та чотири експерименти МЗ («Нове таксі», «Інтерактивна карта», «Бібліотека», «Аптека»).

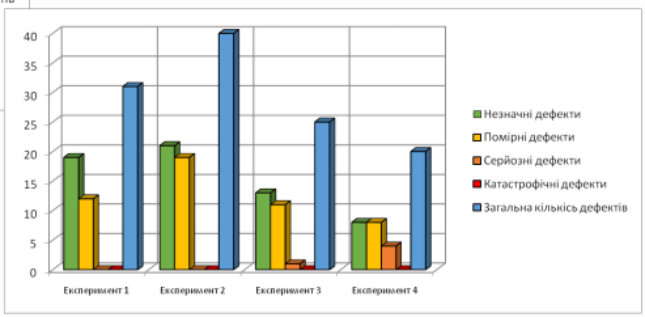


Рисунок 2 – Діаграма виявлених дефектів у МЗ різних рівнів критичності у результаті 4-х проведених експериментів

ЗАГАЛЬНІ ВИСНОВКИ

За результатами виконаних теоретичних та практичних досліджень у роботі розроблено метод та засоби інформаційної технології ідентифікації залишкових дефектів у ПЗ.

У першому розділі проаналізовано існуючі моделі, методи та засоби ідентифікації залишкових дефектів у ПЗ. Виявлено, що до основних моделей для ідентифікації залишкових дефектів відносяться наступні: машинне навчання; статичний аналіз; динамічний аналіз; комбінований аналіз. Також проведений аналіз дозволив виявити тісний взаємозв'язок обраних методів з життєвим циклом дефекту. Це дозволило виявити методи для ідентифікації залишкових дефектів а саме: статичні та дінамічні; рецензування коду; автоматизоване тестування; безперервну інтеграцію та розгортання; контроль версій; рефакторинг. У розділі проаналізовано засоби (системи автоматизованого тестування, моніторинг та журналювання, методи аналізу коду та інші), які можливо використовувати при ідентифікації залишкових дефектів у ПЗ на різних стадіях розробки ПП. В роботі пропонується використовувати автоматизовані засоби тестування.

У другому розділі доведено, що зростання щільності дефектів залежить від кількості рядків програмного коду. Визначено типову щільність дефектів на 1000 рядків коду для ПЗ різного розміру. Встановлено, що мінімальна щільність дефектів при кількості коду менше двох тисяч складає – 1, а при більше 512 тисяч – 5. Відповідно, максимальна щільність дефектів при кількості коду менше двох тисяч складає – 22, а при більше 512 тисяч – 98.

З метою ідентифікації у ПЗ залишкових дефектів з різними ступенями критичності їх наслідків розглянуто інформацію про дефекти, а саме про їх кількість і типи, які були виявлені під час його тестування. Для отримання висновоку про відсутність або наявність залишкових дефектів з різними ступенями критичності їх наслідків, в роботі рекомендовано враховувати взаємний вплив виявлених та залишкових дефектів ПЗ та інші фактори. Для розв'язання такої задачі запропоновано використовувати ШНМ. Для цього розроблено концепцію інформаційної технології ідентифікації залишкових дефектів у ПЗ.

Доведено, якщо відношення сумарного значення дефектів 1-го рівня критичності (незначних дефектів) до загальної кількості виявлених під час базового тестування дефектів дорівнює або перевищує 0.75, то приймається висновок про встановлення наявності залишкових дефектів 2-го рівня критичності (помірних дефектів). Якщо відношення сумарного значення дефектів 2-го рівня критичності (помірних дефектів) до загальної кількості виявлених під час базового тестування дефектів дорівнює або перевищує 0.5, то приймається висновок про встановлення наявності залишкових дефектів 3-го рівня критичності (серйозних дефектів). Якщо кількість дефектів 3-го рівня критичності (серйозних) перевищує 2, то приймається висновок про встановлення наявності залишкових дефектів 4-го рівня критичності (катастрофічних дефектів). Якщо кількість дефектів 4-го рівня критичності (катастрофічних) перевищує або дорівнює 1, то приймається висновок про непридатність програмного забезпечення і можливу відмову системи. Отже, встановлено, що пороги мають наступні значення: 0.75; 0.5; 2; 1. При формуванні бази знань запропоновано створити її з розділу даних і розділу правил.

ЗАГАЛЬНІ ВИСНОВКИ (2)

У третьому розділі вдосконалено нейромережну модель ідентифікації залишкових дефектів у програмному забезпеченні на основі звіту основного тестування, яка відрізняється від відомих тим, що дає можливість враховувати важливість кожного рядка звіту основного тестування, а також взаємний вплив атрибутів в межах дефекту кожного рівня критичності. Вихідні функціонали ШНМ, що відповідають значенням рівнів критичності, дають можливість оцінити сумарний вплив відшуканих під час основного тестування помилок на наявність залишкових дефектів у програмному забезпеченні.

Крім цього, розроблено метод ідентифікації залишкових дефектів у програмному забезпеченні, суть якого полягає у виявленні множини дефектів різних рівнів критичності та аналізу цієї множини на предмет наявності або відсутності залишкових дефектів. Метод відрізняється від відомих тим, що вхідна інформація про результати основного тестування опрацьовується штучною нейронною мережею.

В третьому розділі також розроблено вимоги до інформаційної технології ідентифікації залишкових дефектів у програмному забезпеченні.

У четвертому розділі запропонована інтелектуальна інформаційна технологія ідентифікації залишкових дефектів у ПЗ, яка забезпечує висновок про відсутність залишкових дефектів або про наявність залишкових дефектів з різними ступенями критичності їх наслідків на основі природомовного звіту про дефекти ПЗ, виявлені під час його базового тестування.

Як інструмент запропонованої інформаційної технології запропонована система розробки бездефектного ПЗ шляхом встановлення наявності залишкових дефектів, яка дозволяє користувачу, на основі звіту про результати основного тестування, одержати висновок про відсутність чи наявність залишкових дефектів у програмному забезпеченні (із зазначенням рівня(ів) критичності залишкових дефектів у разі встановлення їх наявності).

У четвертому розділі також апробовано розроблену інформаційну технологію ідентифікації залишкових дефектів у ПП з проведенням десяти експериментів. Для першого експерименту запропонована інтелектуальна інформаційна технологія ідентифікації залишкових дефектів надала висновок про відсутність залишкових дефектів у ПП «Ресторан», оскільки жодне із правил не спрацювало.

У другому експерименті запропонована інтелектуальна інформаційна технологія ідентифікації залишкових дефектів у ПП надала висновок про наявність залишкових дефектів з 3-м та 4-м ступенями критичності їх наслідків у програмній частині кіберфізичної системи «Розумна парковка».

Під час проведення третього експерименту, запропонована інтелектуальна інформаційна технологія ідентифікації залишкових дефектів у ПП надала висновок про наявність залишкових дефектів з 2-м ступенем критичності їх наслідків у програмній частині підсистеми керування освітленням кіберфізичної системи «Розумний будинок».

У четвертому експерименті розроблена система сформувала висновок про встановлення наявності залишкових дефектів 4-го рівня критичності (катастрофічних дефектів) та про непридатність ПЗ і можливу відмову системи «Розумна мийка».

Під час проведення п'ятого експерименту, розроблена система сформувала висновок про встановлення наявності залишкових дефектів 2-го рівня критичності (помірних дефектів) у системі «Флорист».

ЗАГАЛЬНІ ВИСНОВКИ (3)

В останньому, шостому, експерименті розроблена система сформувала висновок про встановлення наявності залишкових дефектів 3-го рівня критичності (серйозних дефектів) у системі «Фільм». Окремо було проведено експерименти застосунків для мобільних телефонів, написаних на мові програмування Kotlin. ПП, які підлягали дослідженню були мобільні застосунки (МЗ): «Нове таксі», «Інтерактивна карта», «Бібліотека» та «Аптека». Для першого, другого і третього експериментів запропонована інтелектуальна інформаційна технологія ідентифікації залишкових дефектів надала висновок про відсутність залишкових дефектів у МЗ «Нове таксі», «Інтерактивна карта» і «Бібліотека».

Для четвертого експерименту розроблена система сформувала висновок про встановлення наявності залишкових дефектів 4-го рівня критичності (катастрофічних дефектів) у МЗ «Аптека». Як показали проведені дослідження, розроблена система розробки бездефектного ПЗ шляхом встановлення наявності залишкових дефектів в результаті свого функціювання видає висновок про встановлення наявності або відсутності залишкових дефектів (із зазначенням рівня(ів) критичності залишкових дефектів у разі встановлення їх наявності) на основі опрацювання інформації про кількість і типи дефектів, виявлених під час базового тестування, що міститься у звіті базового тестування.

Тому, запропонована програма дозволяє розробляти бездефектне ПЗ виявлення залишкових дефектів у програмах після базового тестування, за рахунок чого відбувається підвищення достовірності тестування і відповідно зростання якості ПЗ.

Наукова новизна отриманих результатів:

– набув подальшого розвитку метод ідентифікації залишкових дефектів у програмному забезпеченні, який відрізняється від відомих тим, що вхідна інформація про результати основного тестування обробляється штучною нейронной мережею (ШНМ), і забезпечує визначення множини дефектів різного рівня критичності та аналіз цієї множини на наявність або відсутність залишкових дефектів;

– набула подальшого розвитку інформаційна технологія ідентифікації залишкових дефектів у програмних продуктах за рахунок опрацювання природомовних звітів про результати основного тестування, яка забезпечує висновок про відсутність або наявність залишкових дефектів у аналізованому програмному забезпеченні.

Практична значущість отриманих результатів полягає у проєктуванні та реалізації інтелектуальної інформаційної технології ідентифікації залишкових дефектів у програмних продуктах, яка забезпечує висновок про відсутність залишкових дефектів або про наявність залишкових дефектів з різними ступенями критичності їх наслідків на основі природомовного звіту про дефекти програмного продукту, виявлені під час його основного тестування.

За темою кваліфікаційної роботи опубліковано одну статтю у матеріалах конференції, що індексуються в наукометричній базі Scopus, і одну статтю у фаховому виданні України:

1) I. Zasornova, T. Hovorushchenko, M. Fedula, V. Buzyl. Intelligent Information Technology for Identification of Remaining Defects in Software Products // Proceedings of the 2023 IEEE 12-th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS-2023, Dortmund, 7-9 September 2023), vol. 1, pp. 33-37;

2) I. Zasornova, T. Hovorushchenko, O. Voichur. Study of Software Testing Tools According to the Testing Levels. Computer Systems & Information Technologies. 2023. №1. Pp. 38-46.

UNICHECK
by Turnitin

Ім'я користувача:
Кафедра КІ

ID перевірки:
1015957729

Дата перевірки:
01.12.2023 09:28:43 EET

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
01.12.2023 11:05:19 EET

ID користувача:
100005591

Назва документа: Засорнова_Метод та засоби інформаційної технології ідентифікації залишкових дефектів...

Кількість сторінок: 97 Кількість слів: 18574 Кількість символів: 136732 Розмір файлу: 1.76 MB ID файлу: 1015634406

10.2%
Схожість

Найбільша схожість: 6.41% з джерелом з Бібліотеки (ID файлу: 1008424520)

9.84% Джерела з Інтернету 219 ..... Сторінка 99

8.11% Джерела з Бібліотеки 63 ..... Сторінка 103

0.09% Цитат

Цитати 13 ..... Сторінка 104

Посилання 1 ..... Сторінка 104

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи 11

Fri Dec 01 12:02:12 EET 2023, Медзатий Дмитро Миколайович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 0.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 10%**

	Документ		Сумарний збіг по Базі Даних	
ID: 121566 Назва: ДП Метод та засоби інформаційної технології ідентифікації залишкових дефектів у програмному забезпеченні Додано в БД: 2023-12-01 Автора: Засорнова І.О. Керівники: Говорущенко Т.О. Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	118548	777	1093 (1%)	16 (2%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник: Засорнова Ірина Олександрівна

Тема: Метод та засоби інформаційної технології ідентифікації залишкових дефектів у програмному забезпеченні

Спеціальність: 126 «Інформаційні системи та технології»

Обсяг дипломної роботи:

Кількість сторінок записки 138 с.

1. Короткий зміст роботи та прийнятих рішень: Метою кваліфікаційної роботи є ідентифікація залишкових дефектів у програмному забезпеченні (ПЗ) шляхом розробки методу та засобів інформаційної технології. Вивчаючи ці методи та засоби, можливо отримати уявлення про їх переваги та обмеження, а також зрозуміти їхню придатність для задач виявлення дефектів. Ці дослідження сприятимуть покращенню загального процесу розробки ПЗ.

2. Висновок про відповідність роботи дипломному завданню: Робота повністю відповідає поставленому завданню.

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: В першому розділі проведено аналіз відомих моделей, методів та засобів ідентифікації залишкових дефектів програмного забезпечення. В другому розділі представлено концепцію інформаційної технології ідентифікації залишкових дефектів у програмному забезпеченні, проведено препроцесінг та опрацювання даних для ідентифікації залишкових дефектів у ПЗ, описано інформаційні потоки в процесі ідентифікації залишкових дефектів ПЗ. В третьому розділі розроблено алгоритм, модель і метод ідентифікації залишкових дефектів у ПЗ, розроблено також вимоги до інформаційної технології ідентифікації залишкових дефектів у ПЗ. Набув подальшого розвитку метод ідентифікації залишкових дефектів у програмному забезпеченні, який відрізняється від відомих тим, що вхідна інформація про результати базового тестування обробляється штучною нейронною мережею (ШНМ), і забезпечує визначення множини дефектів різного рівня критичності та аналіз цієї множини на наявність або відсутність залишкових дефектів. В четвертому розділі розроблено інформаційну технологію ідентифікації залишкових

дефектів у ПЗ, описано структуру такої інформаційної технології та результати її функціонування. Набула подальшого розвитку інформаційна технологія ідентифікації залишкових дефектів у програмних продуктах за рахунок опрацювання природомовних звітів про результати основного тестування, яка забезпечує висновок про відсутність або наявність залишкових дефектів у аналізованому програмному забезпеченні. Практична значущість отриманих результатів полягає у проєктуванні та реалізації інтелектуальної інформаційної технології ідентифікації залишкових дефектів у програмних продуктах, яка забезпечує висновок про відсутність залишкових дефектів або про наявність залишкових дефектів з різними ступенями критичності їх наслідків на основі природомовного звіту про дефекти програмного продукту, виявлені під час його основного тестування.

4. Позитивні сторони роботи: отримання двох пунктів наукової новизни.

5. Негативні сторони роботи:

6. Оцінка графічного оформлення та пояснювальної записки роботи: Пояснювальна записка оформлена коректно, згідно з діючими стандартами оформлення документації.

7. Відгук про роботу в цілому: Робота виконана на високому науково-технічному рівні.

8. Інші зауваження: _____

9. Оцінка дипломної роботи: відмінно.

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) _____

Мартинюк Валерій Володимирович,
зав. каф. АКІТтаР, ХНУ

"15" грудня 2023 р. _____ (підпис)

Завідувачу кафедри КІІС
д-р.техн.наук, проф. Говорущенко Т.О.

Засорнова Ірина Олександрівна
ПІБ здобувача вищої освіти

ФІТ, 2 курсу, групи ІСТм-22-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлена. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщена та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

1 грудня 2023 року

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОМАЦІЙНИХ СИСТЕМ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Метод та засоби інформаційної технології ідентифікації залишкових дефектів у програмному забезпеченні

Автор: Засорнова Ірина Олександрівна

Спеціальність: 126 – Інформаційні системи та технології

Освітня програма: Інформаційні системи та технології

Науковий керівник: Говорущенко Тетяна Олександрівна, д.т.н, професор

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) запозичення розміщені в розділах є збігом із статтями автора: 1) I. Zasornova, T. Hovorushchenko, M. Fedula, V. Buzyl. Intelligent Information Technology for Identification of Remaining Defects in Software Products // Proceedings of the 2023 IEEE 12-th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS-2023, Dortmund, 7-9 September 2023), vol. 1, pp. 33-37; 2) I. Zasornova, T. Hovorushchenko, O. Voichur. Study of Software Testing Tools According to the Testing Levels. Computer Systems & Information Technologies. 2023. No1. Pp. 38-46;
2) усі запозичення фрагментарні, або мають належним чином оформленні посилання;
3) окремі виявлені збіги є загальновживаними фразами або виразами, про що свідчить посилання системи на збіг з джерелами на один фрагмент речення;
4) в якості запозичень в окремих місцях системою зафіксовано послідовності чотирьохрозряних двійкових кодів, які є вхідними даними до великої кількості задач і не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;
5) всі зафіксовані системою ознаки модифікації тексту відносяться до комбінування латинських символів зі україномовними скороченнями індексів в формулах, що не є модифікацією тексту. Або умовних скорочень слів.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості Unicheck, складає 10.2% і адресується до 282 першоджерела; та системою Anti-Plagiarism складає 0%, що, з урахуванням наведених обгрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ Т. О. Говорущенко

Гарант ОП _____ О. О. Павлова

Завідувач кафедри КІС _____ Т. О. Говорущенко