Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Навчально-науковий центр заочної форми навчання

Кафедра Програмної інженерії

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка
рівень вищої освіти – другий (магістерський)

Дослідження методів нейромережевих технологій при проведенні функціонального тестування програмного забезпечення

Виконав: студент 2 курсу,
групи ПЗСзм-19-1
Лановий О.Ф.
спеціальності: 121 Інженерія програмного забезпечення
освітньо-професійної програми:
Програмне забезпечення систем

Керівник атестаційної роботи
проф. Дудар З.В.

Допускається до захисту
Зав. кафедри, проф. _____ З.В.Дудар

Харків 2020

Харківський національний університет радіоелектроніки

Факультет Навчально-науковий центр заочної форми навчання

Кафедра Програмної інженерії

Рівень вищої освіти – другий (магістерський)

Спеціальність 121 Інженерія програмного забезпечення
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Програмне забезпечення систем

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)
«____»_____ 20 ___ р.

ЗАВДАННЯ

НА АТЕСТАЦІЙНУ РОБОТУ

студентові Лановому Олексію Феліксовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження методів нейромережевих технологій при проведенні функціонального тестування програмного забезпечення
затверджена наказом університету від "26" 10 2020 р. № 174Ст

2. Термін подання студентом роботи до екзаменаційної комісії
10 грудня 2020 р.

3. Вихідні дані до роботи *електронні ресурси за обраною тематикою, методи функціонального тестування програмного забезпечення, алгоритми штучних нейронних мереж, алгоритми класифікації та порівняння*

4. Перелік питань, що потрібно опрацювати в роботі *аналіз проблемної галузі і постановка задачі, огляд існуючих методів функціонального тестування, використання методів та алгоритмів штучних нейронних мереж в тестуванні програмного забезпечення, опис розробленої методики тестування програмної системи з використанням нейромережевих технологій.*

5. Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім’я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Спецчастина	проф. Дудар З.В.		8.12.2020

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Аналіз проблемної області та постановка задачі	27.10.2020-08.11.2020	виконано
2	Дослідження методів ШІ та їх використання в тестуванні ПЗ	08.11.2020-11.11.2020	виконано
3	Розробка моделі взаємодії даних	08.11.2020-20.11.2020	виконано
4	Розробка структури ШНМ	15.11.2020-25.11.2020	виконано
5	Нормалізація даних та проведення експериментів	25.11.2020-30.11.2020	виконано
6	Аналіз та оформлення результатів, підготовка пояснювальної записки	30.11.2020-07.12.2020	виконано
7	Підготовка презентації та доповіді	05.12.2020-08.12.2020	виконано
8	Перевірка на плагіат	08.12.2020	виконано
9	Попередній захист	10.12.2020	виконано
10	Нормоконтроль, рецензування	11.12.2020	виконано
11	Занесення диплома в електронний архів	12.12.2020	виконано
12	Допуск до захисту у зав. кафедри	15.12.2020	виконано

Дата видачі завдання 26 жовтня 2020 р.

Студент _____ Лановий О.Ф.
(підпис)

Керівник роботи _____ проф. Дудар З.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Атестаційна робота магістра містить: 76 с., 42 рис., 11 табл., 40 джерел.

ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ТЕСТОВИЙ ВИПАДОК, ШТУЧНИЙ ІНТЕЛЕКТ, МАШИННЕ НАВЧАННЯ, ШТУЧНА НЕЙРОННА МЕРЕЖА, ГЕНЕТИЧНИЙ АЛГОРИТМ.

Об’єкт – дослідження нейромережевого підходу при проведенні функціонального тестування програмного забезпечення.

Мета роботи – вдосконалення методів тестування програмного забезпечення з використанням підходів машинного навчання та алгоритмів штучного інтелекту.

Методи дослідження. В роботі використовувалися методи машинного навчання та кластеризації, моделювання для розробки математичної моделі вибору найкращого методу для обробки вибірок на підставі характеристик вхідних даних.

SOFTWARE TESTING, TEST PATTERN, ARTIFICIAL INTELLIGENCE, MACHINE LEARNING, ARTIFICIAL NEURAL NETWORK, GENETIC ALGORITHM.

Object – the researching of neural network approach to black-box testing software.

Purpose – to explore the concept of improving methods of software testing approaches using machine learning algorithms and artificial intelligence.

Research methods. The paper used various methods of machine learning and clustering, modeling to develop a mathematical model for selecting the best method for processing samples based on the characteristics of the input data.

СКОРОЧЕННЯ

ГА	– генетичний алгоритм
МН	– машинне навчання
НзП	– навчання з підкріпленням
ПЗ	– програмне забезпечення
ШІ	– штучний інтелект
ШНМ	– штучна нейронна мережа

ЗМІСТ

ВСТУП…… 6
1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ……...…… 8
1.1 Підходи та методи тестування програмних продуктів …… 8
1.2 Аналіз застосування методів ШІ для тестування ПЗ…… 14
1.3 Алгоритми ШІ …… 17
1.4 Застосування методів ШІ в методологіях DevOps і Agile…… 21
1.5 Постановка задачі дослідження …… 22
2 ЗАСТОСУВАННЯ МЕТОДІВ ШІ ДЛЯ ТЕСТУВАННЯ ПЗ ……... 25
2.1 Огляд опублікованих результатів наукових досліджень …… 25
2.2 Використання алгоритмів ШНМ в циклі розробки ПЗ …… 27
2.3 Методика застосування ШНМ в тестуванні ПЗ …… 31
3 ОПИС МЕТОДУ ДОСЛІДЖЕННЯ ……... 38
3.1 Модель тестування ПЗ …… 38
3.2 Тестове покриття . …… 40
3.3 Інтелектуальний фазінг …… 41
3.4 Навчання інтелектуального агента …… 43
3.5 Методика проведення досліджень . …… 45
4 ПРОВЕДЕННЯ ЕКСПЕРИМЕНТІВ ТА АНАЛІЗ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ …… 47
4.1 Підготовка вхідних даних …… 47
4.2 Проведення експериментів …… 53
ВИСНОВКИ …… 58
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ …… 60
ДОДАТОК А. Перелік наукових праць співробітників кафедри ПІ за темою дослідження …… 65
ДОДАТОК Б. Слайди презентації …… 66

ВСТУП

Вплив розвитку глобальної мережі Інтернет на людство не має історичних аналогів. Фактично це початок епохи електронного проникнення в усі сфери людського життя. Сучасна людина користується мережею Інтернет як найбільш оперативним джерелом отримання інформації.

За останній час значно зросли обсяги розробки програмного забезпечення, що обумовлює необхідність в належному проведенні його тестування. Тестування програмного забезпечення являє собою процес дослідження того, наскільки реальна поведінка системи відповідає вимогам і очікуванням. Тести, які використовуються для цих перевірок, ґрунтуються на задокументованих вимогах замовника. Необхідно максимально підвищити ймовірність того, що тестований продукт буде працювати як належить за будь-яких обставин і буде відповідати всім описаним вимогам.

Штучний інтелект (ШІ) стає все більш важливою частиною IT-бізнесу. Разом з цим слід зазначити, що саме для тестування програмного забезпечення широко цей метод не використовується. Враховуючи той факт, що складність програмного забезпечення значно зросла з розвитком технологій програмування, тестування програмного забезпечення не відчуло значних змін. Тестування програмного забезпечення (ПЗ), як і раніше, виконується в значній мірі з використанням тих самих методів, що і десять років тому, хоча необхідність в прискоренні темпів тестуванні зросла.

Штучний інтелект та машинне навчання (МН) можуть бути використані у якості одного з підходів до скорочення ручної роботи при тестуванні ПЗ, а також при традиційній автоматизації тестування, яка, як і раніше, вимагає значного обсягу ручної роботи. Слід зазначити, що не зважаючи на стрімкий розвиток, технології ШІ та МН залишаються невідомими для більшої частини спільноти розробників програмного забезпечення.

В рамках написання атестаційної роботи розглядаються питання щодо вдосконалення методів тестування програмного забезпечення за рахунок

використання методів штучного інтелекту. В роботі досліджуються різні підходи до тестування програмного забезпечення з використанням ШІ, який починає грати все більш важливу роль, знаходячи своє застосування в інструментах тестування, розроблених протягом останніх кількох років. Основна мета дослідження полягає у формуванні рекомендацій з використання методів ШІ для тестування програмного забезпечення.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Підходи та методи тестування програмних продуктів

Вдосконалення прийомів і методів тестування сучасних програмних продуктів є фокусом пильної уваги з боку компаній, що займаються розробкою ПЗ, у зв'язку із незаперечною важливістю забезпечення належного рівня якості розробленого ПЗ. Тестування, що є основним способом контролю якості програмних рішень, може бути визначено як процес виконання програми з метою перевірки відповідності між її реальною та очікуваною поведінкою. Тестування проводять на кінцевому наборі тестових випадків, визначених певним способом. В найбільш загальному випадку процес тестування може бути представлений сукупністю чотирьох основних активностей: планування робіт, розробка тестів, їх виконання і аналіз результатів перевірки ПЗ [1].

Є кілька основних підходів до організації процесу тестування ПЗ, які формують режим проведення тестування. Ці методи зазвичай вважаються самодостатніми для пошуку помилок і багів в усій системі.

Методика тестування без будь-яких знань про внутрішню структуру додатка отримала назву «чорного ящика» (Black-box testing). Тестер не звертає увагу на архітектуру системи і не має доступу до вихідного коду. Як правило, при виконанні тестування тестер буде взаємодіяти з інтерфейсом системи у якості користувача, надаючи вхідні дані і аналізуючи виходи, не знаючи, як і де обробляються входи [2].

Тестування методом «білого ящика» (White-box testing) – це детальне дослідження внутрішньої логіки і структури коду. Тестування методом «білого ящика» також називається відкритим тестуванням. Для повноцінного проведення процедури тестування тестер повинен знати внутрішню структуру програми та роботу коду.

Метод «сірого ящика» не передбачає від тестера знання вихідного коду, замість цього вони повинні знати особливості інтерфейсу і функціональні специфікації. Цей метод тестування програми з обмеженим знанням її внутрішньої структури, при

цьому ключовою для тестера є фраза – «чим більше ви знаєте, тим краще». Фактично цей метод пропонує комбіновані переваги тестування методами «чорного ящика» і «білого ящика», де це можливо. Це є передумовою того, що тестер може підготувати кращі тестові дані і сценарії тестування при складанні плану тестування [1].

Протягом життєвого циклу розробки і супроводу програмного забезпечення тестування здійснюється на різних його рівнях. Рівень визначає, який об'єкт або елемент програмного продукту розглядається в ході перевірки. Прийнято виділяти чотири типових рівня тестування (рис. 1.1) [3].

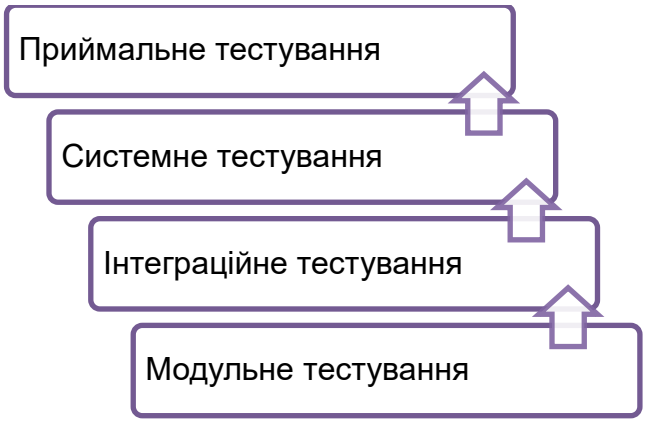

Рис. 1.1 – Рівні тестування ПЗ

Модульне тестування розглядає окремий блок програми або програми як найменшу функціональну їх частину, не здатну функціонувати окремо, а тільки в сукупності з іншими модулями. Таким чином, модульне тестування передбачає випробування програми на об'єктному рівні.

Тестування на інтеграційному рівні служить для перевірки взаємодії декількох модулів в зв'язці один з одним. Найбільш поширені такі покрокові підходи інтеграційного тестування як «Зверху вниз» (рис. 1.2) і «Знизу вгору» (рис. 1.3).

Якщо для перевірки працездатності ПЗ використовується тестування «Зверху вниз», то в першу чергу розглядаються високорівневі модулі програми, а на подальших кроках – модулі, що розташовуються на більш низьких рівнях архітектурної схеми програми. Для проведення коректного тестування модуля підключаються допоміжні класи-заглушки, що виконують функцію заміщення

реальних класів рівнем нижче розглянутого. У міру готовності класи-заглушки замінюються реальними активними компонентами. Такий підхід дозволяє отримати уявлення про структуру програми на проміжних етапах тестування, оцінити організацію взаємодії продукту з користувачем [4].

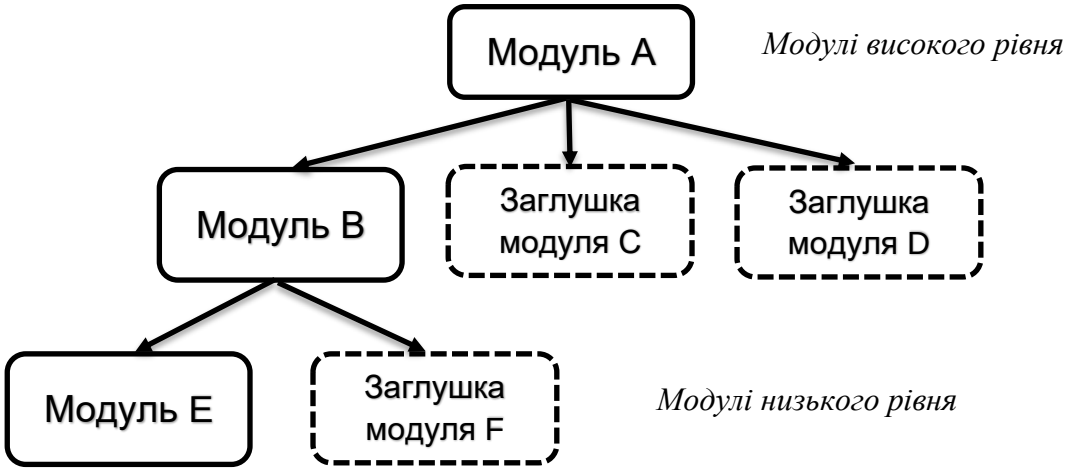

Рис.1.2 – Тестування «Зверху вниз»

У разі проведення тестування «Знизу вгору» в першу чергу розглядаються так звані термінальні класи – класи, які не використовують у своїй роботі методи інших класів. Для тестування обирають такі модулі, які під час свого функціонування використовують методи тільки вже протестованих класів. Важливу роль відведено драйверу кожного з модулів. Драйвер містить в собі фіксовані тестові дані, ініціалізує тестовий модуль, отримує результати тестування та порівнює їх з очікуваним результатом роботи модуля.

Недоліком використання цього підходу у тестуванні ПЗ є відсутність структури програми на проміжних етапах тестування. Разом з тим цей підхід забезпечує більш просту перевірку результатів за рахунок відсутності необхідності використання заглушки для незавершених модулів. Цей підхід найкращим чином підходить для ситуацій, коли помилки у своїй більшості зосереджені в модулі нижнього рівня.

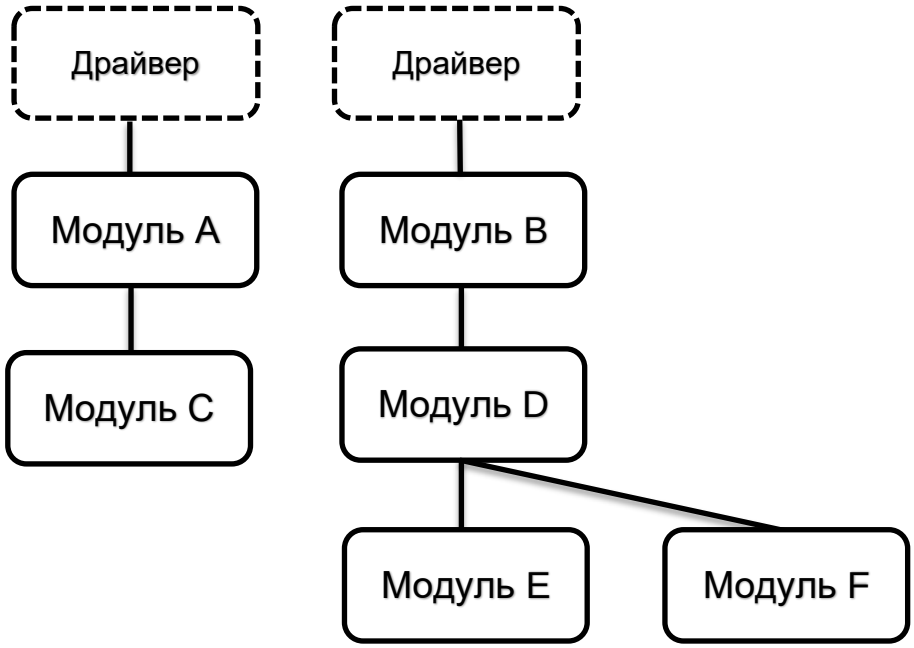

Рис.1.3 – Тестування «Знизу вгору»

На етапі системного тестування весь продукт перевіряється на наявність помилок в функціонуванні при сполученні як програмних, так і апаратних компонентів системи. На даному рівні тестується правильність використання ресурсів системи, сумісність з оточенням, повнота і коректність функціональності, зручність використання. Для виконання системного тестування формуються тестові випадки на підставі пропонованих вимог щодо розроблюваного продукту (технічного завдання) і сценаріїв використання системи.

Приймальне тестування здійснюється з метою визначення готовності передачі програмного продукту замовнику. Приймальні випробування можуть проводитися як з боку фахівців компанії, що розробляють ПЗ, так і з боку клієнта (в останньому випадку мається на увазі зовнішнє тестування).

Крім рівнів тестування прийнято виділяти підходи до виконання тестування. Розрізняють ручне тестування та автоматичне виконання тестових сценаріїв [1].

Проведення ручного тестування визначає дії фахівця з тестування як якби він був користувачем системи, що розробляється. Ручне тестування здійснюється шляхом моделювання можливих сценаріїв активності користувача програмного продукту. Процедура, яка описує методику виконання тестування, наступна: визначення набору та порядку виконання певних тестових випадків, діапазону

значень вхідних параметрів, очікуваних результатів поведінки системи в кожному конкретному випадку тощо. Все складається заздалегідь і документується. Обов'язковою умовою успішного проведення тестування є чітке розуміння з боку фахівця всіх пунктів зазначеної процедури. Детальність опису тестового випадку може бути різною, в зв'язку з чим від фахівця з тестування потрібна гнучкість у визначенні вхідних даних та при оцінці нечітко сформульованих критеріїв результатів роботи програми.

Автоматизоване тестування є аналогом ручного, проте для виконання тестів і оцінки їх результатів використовуються програмні засоби. За допомогою спеціалізованих програмних інструментів здійснюється запуск тестів, ініціалізація, виконання тестового сценарію, подальший аналіз та видача результатів тестування. Автоматизація тестування дозволяє виключити вплив людського фактору на якість тестування, забезпечити сувору одноманітність виконання тестів, гарантувати виявлення помилок при виконанні програми і чітке дотримання кроків алгоритму кожного тестового випадку. Разом з цим автоматизація процесу тестування значно скорочує часові витрати, виключає потреби в звірці з технічним завданням та інструкціями, що характерно для ручного тестування. Слід зазначити, що звіти про результати тестування зберігаються і можуть бути відправлені всім зацікавленим особам в автоматичному режимі.

Значною перевагою є і той факт, що виконання тестів відбувається без втручання фахівця з тестування і не залежить від графіка роботи фахівців.

Виділяють три рівні автоматизації тестування: рівень модульного тестування, рівень функціонального тестування та рівень тестування через призначений для користувача інтерфейс [5].

Кожен рівень характеризується власним об'єктом тестування, і тому для забезпечення максимальної якості розробки програмного продукту необхідно виконувати автоматичні тести всіх рівнів.

В процесі модульного тестування здійснюється автоматизована перевірка працездатності окремих модулів або методів вихідного коду програми в ізоляції від решти частин коду. Як правило, модульні тести створюються і виконуються самими

розробниками програми. Як інструмент перевірки виступають unit-тести, які досить часто пишуться завчасно, до написання коду програми, з метою перевірки коректність його роботи.

Автоматизація на функціональному рівні охоплює перевірку логіки роботи програми, яка не може бути протестована через призначений для користувача інтерфейс ПЗ. З цієї причини для проведення автоматизованого тестування на функціональному рівні фахівцям групи контролю якості надається доступ до функціонального шару продукту, що дозволяє перевірити логіку роботи програми в автоматичному режимі, минаючи використання призначеного для користувача інтерфейсу.

Рівень автоматизації тестування через призначений для користувача інтерфейс, який також називається GUI-автоматизацією, є найбільш затребуваним, оскільки він імітує прямі дії користувача і не вимагає доступу до вихідного коду програми.

Однією з технік GUI-автоматизації є написання сценарію – послідовності дій, що виникають під час взаємодії користувача з системою. Створення тестів при такій формі автоматизації має на увазі під собою програмування на мовах, спеціально розроблених для зазначеної мети, які забезпечують пом'якшення основних проблем інструментів відтворення і запису.

Розроблені поведінкові скрипти (сценарії) застосовуються як для окремих частин програмного забезпечення, так і всієї системи в цілому. Такі скрипти симулюють роботу користувача системи.

Поряд з усіма очевидними перевагами використання автоматизованого тестування, такими як висока швидкість виконання, висока повторюваність, надійність і незалежність від довжини тестового циклу, слід зазначити, що автоматичні методи тестування в багатьох ситуаціях поступаються ручному тестуванню. Це пов'язано з двома основними проблемами: складністю написання автоматичних тестів, нелінійною залежністю від складності програмного забезпечення, відсутністю випадковості під час проведення тестів, відсутністю тієї гнучкості, яку можуть забезпечити фахівці при ручному методі тестування. Для усунення зазначених недоліків необхідно вирішити дві задачі: спростити процедуру

створення автотестів або створювати тестове покриття в автоматичному режимі і додати випадковість в поведінку скриптів, що виконуються за чітким алгоритмом [6].

Для вирішення зазначених задач може бути запропонований метод, який базується на використанні штучних нейронних мереж (ШНМ). Використання нейронних мереж показує більш ніж гідні результати в різних додатках. Оскільки програмне забезпечення може бути розглянуто в якості опції в багатовимірному просторі, то для цієї моделі також можна застосовувати методи нейронних мереж.

1.2 Аналіз застосування методів ШІ для тестування ПЗ

Тестування програмного забезпечення є основним способом перевірки відповідності програмного забезпечення певним вимогам і становить близько половини вартості часу розробки. Сучасні дослідження, що проводяться з метою вдосконалення якості тестування програмного забезпечення, дозволяють стверджувати, що поліпшення в системній інфраструктурі проведення процесу тестування можуть заощадити до третини його витрат [7]. З іншого боку, в останні десятиріччя концепції ШІ та МН успішно використовувалися для дослідження можливостей опрацювання даних в різних областях [8]. МН використовується для навчання машин для більш ефективної обробки даних за рахунок моделювання концепцію навчання раціональних істот, і може бути реалізований за допомогою алгоритмів (або методів) ШІ, що відображають парадигми вибору раціональних характеристик, такі як генетичні, статистичні та ймовірностні. За допомогою алгоритмів ШІ і на основі підходу машинного навчання можна досліджувати і отримувати інформацію для класифікації, зв'язування, оптимізації, кластеризації, прогнозування, виявлення закономірностей проведення тестування програмного забезпечення.

Відомо, що методи ШІ використовуються для побудови прогнозних моделей для досягнення різних інженерних цілей [9], але це не знаходить свого впровадження

для підтримки систем тестування. Здається логічним застосовувати ШІ при тестуванні програмного забезпечення, однак відсутність оракула (механізму, який розрізняє правильне і неправильне поводження систем тестування) в даний час є вузьким місцем. ШІ нечасто використовується для виявлення помилок в системах тестування через відсутність автоматизації визначення оракулу. Винятком є регресійні тестування, при якому правильна робота системи тестування може бути отримана оракулом на підставі результатів тестування попередньої версії [10].

Програмне забезпечення за своєю суттю представляє функцію перетворення вхідної інформації у вихідну в багатовимірному просторі. В такий математичній моделі досить легко застосувати нечіткі логічні методи, тобто нейронні мережі.

Нейронна мережа – це математична модель, заснована на когнітивних процесах мозку. Так само, як і мозок, нейронна мережа складається з вузлів (нейронів) і змінних зв'язків між ними.

Нейрони поділяються на 3 групи за їх властивостями: вхідні нейрони, вихідні нейрони та нейрони прихованого шару (обчислювальні вузли). Фактично нейронна мережа – це особливий спосіб визначення функції [11].

Процес навчання нейронної мережі досить простий. Готується навчальний зразок – набір пар вхідних та вихідних векторів із відомими та правильними значеннями. Елементи цього набору задовольняють наступній властивості – незалежності, отже, вибірку можна розширити, а її елементи переставити в будь-якому порядку. Навчальний зразок передається через нейронну мережу, і в результаті цього процесу ваги нейронних ланок змінюються таким чином, щоб визначити функцію, яка задовольняє навчальній вибірці. Очевидно, що якість роботи нейронної мережі залежить від якості її підготовки.

Тестовий зразок готується подібно до навчального, але з різними значеннями. Якщо результат тесту є незадовільним, то необхідно або навчити мережу, або змінити її структуру.

Добре навчена нейронна мережа здатна передбачити результат за допомогою набору вхідних значень. Разом з цим додатково він вирішує проблему класифікації –

якщо класи у вхідному наборі визначені заздалегідь, або проблему кластеризації, тобто ідентифікацію класів у наборі.

Багатошарова нейронна мережа навчається на оригінальному програмному додатку за допомогою випадково сформованих тестових даних, що відповідають специфікації. Нейронна мережа може бути навчена з розумною точністю опрацьовувати оригінальну програму, хоча вона може бути не в змозі класифікувати дані тесту на 100 відсотків правильно. Фактично навчена нейронна мережа стає імітаційною моделлю програмного додатку. Коли створюються нові версії оригінальної програми встає потреба у регресійному тестуванні. Змінений код виконується на тестових даних, а отримані результати порівнюються з результатами роботи нейронної мережі. Нові версії не змінюють існуючі функції, а це означає, що програма повинна генерувати однакові результати для тих самих входів. Потім інструмент порівняння приймає рішення про те, чи є результат тестування програми позитивним чи негативним виходячи з функцій активації мережі.

Методи МН можна класифікувати за трьома основними категоріями [12], але для кожного з підходів машинного навчання існує один або кілька алгоритмів ШІ, що реалізують відповідний метод навчання:

– контрольоване навчання (або навчання з вчителем) – для формування прогнозу на основі заданого набору зразків навчальних даних спочатку даються правильні відповіді для цільової змінної, а задачею алгоритму ШІ є спроба відтворити правильну відповідь на основі визначеного набору даних. Цей тип навчання використовується для цілей класифікації і регресії об'єктів при мінімізації помилок виведення. Набір даних можна розділити на навчальні дані і дані тестування, що дозволяє обчислити точність і втрати конкретного алгоритму;

– неконтрольоване навчання (або навчання без вчителя) – цей метод використовується для виявлення прихованих закономірностей в даних, які не мають цільової змінної (очікуваних результатів), і організовує дані в групу кластерів для опису їх структури. При навчанні без учителя системі не надаються навчальні дані, що зазвичай є більш складним завданням, тому його частіше використовують для дослідження даних [13]. Алгоритми, які зазвичай використовуються для цього

методу, можуть бути k-середніми (де користувач вказує бажану кількість кластерів) або мати ієрархічну кластеризацію (де користувачеві не потрібно фіксувати певну кількість кластерів), серед яких намагаються згрупувати об'єкти на підставі їх подібності [14];

– навчання з підкріпленням – алгоритм постійно адаптується до навколишнього середовища на основі зворотного зв'язку, який може бути позитивним або негативним [11]. Алгоритм обирає дію для кожного елементу набору даних, а потім дізнається про ефективність отриманого рішення, змінюючи свою стратегію для кращого навчання і отримання привілеїв за виконані дії [5].

Значна кількість алгоритмів ШІ, які можуть бути використані при тестуванні програмного забезпечення, призводить нас до необхідності зробити вибір серед них. Аналіз проведених досліджень, що знайшов своє відображення в переліку першоджерел посилання, свідчить про те, що найпоширеними методами використання алгоритмів ШІ стали штучні нейронні мережі (включаючи опорні векторні машини – SVM), підхід з використанням k-найближчого сусіда (kNN), наївний байесовский класифікатор, дерева рішень і правил, індукційний алгоритм тощо. Розглянемо деякі з цих методів більш детально.

1.3 Алгоритми ШІ

1.3.1 Штучна нейронна мережа. Штучні нейронні мережі – це апроксиматори функцій, які зазвичай використовуються для апроксимації безперервних станів і дій. Як правило, вони краще підходять для класифікації великих просторових станів, які інакше не могли б бути оброблені [11]. Вони складаються з вхідного шару (з вхідними даними), вихідного шару (здатного класифікувати дані) і одного або декількох прихованих шарів, що обробляють інформацію. Проста ШНС має тільки один прихований шар, в той час як глибока ШНС має їх два або більше (рис. 1.4).

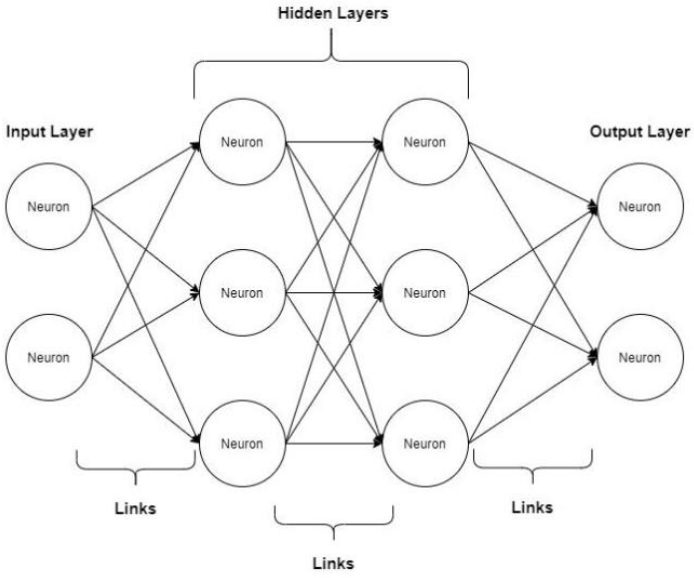

Рис.1.4 – Глибока штучна нейронна мережа

Одиницею ШНМ є нейрон, а зв'язки між нейронами називаються зв'язками або синапсами. Нейрон приймає кілька входів $x_1, x_2, ..., x_n \in \mathbb{R}$ і на вихід видає $o \in \mathbb{R}$.

Найбільш поширена конструкція нейрона – персептрон, який може бути представлений двома рівняннями: передавальної функцією і функцією активації [12]. Передавальна функція виглядає наступним чином:

$$z(x_i)=b+\sum_{i=1}^{n} w_i x_i \qquad (1)$$

де b – константа зсуву, яку можна використовувати для затримки спрацьовування функції активації (що допомагає створити кращу модель), а w_i представляє собою ваговий коефіцієнт, який призначається входу.

Є кілька типів функцій активації. Одна з найпоширеніших –функція активації бінарного кроку:

$$o(z)=\begin{cases} 1, z>0 \\ -1, z\leq 0 \end{cases} \qquad (2)$$

У цьому випадку функція активації формує 1 або -1 в залежності від порогу спрацювання (в даному випадку – 0).

Вага для кожного перцептрону Δw_i обчислюється як випадкове припущення і оновлюється у відповідності до наступного правила:

$$\Delta w_i = \eta(t-o)x_i \qquad (3)$$

де η – швидкість навчання (позитивна константа), t – цільової вихід, o – поточний вихід для входу. Якщо t та o дорівнюють один одному, досягається бажаний результат, і вага більше не оновлюється [11].

1.3.2 Генетичний алгоритм. Генетичний алгоритм (ГА) – це стратегія оптимізації, яка імітує принципи дарвінівської теорії еволюції. ГА зазвичай використовується для вирішення нелінійних, мультимодальних і розривних завдань оптимізації [11]. В алгоритмі повинні бути визначені дві речі: функція придатності, здатна оцінювати якість рішення проблеми, і спосіб представлення «ДНК» можливих рішень.

Робочий процес ГА (рис.1.5) починається з ініціалізації можливих рішень, а потім оцінки цих рішень стають функцією придатності. Якщо критерії завершення дотримані, буде обрано саме оптимальне рішення. В противному випадку для модифікації відбираються більш перспективні кандидати.

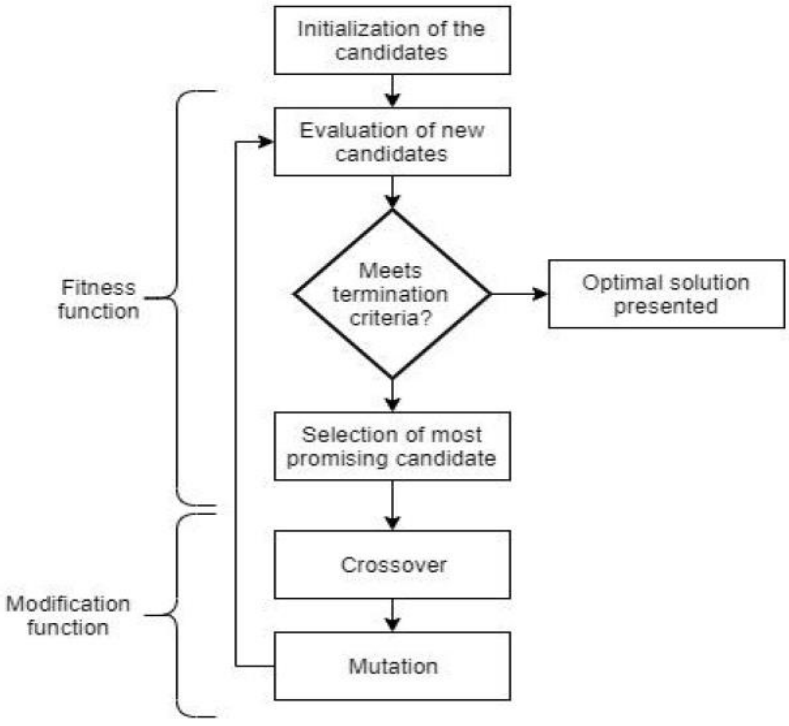

Рис. 1.5 – Генетичний алгоритм

Зміна кандидатів включає кросовер, а потім мутацію. Кросовер генерує нових кандидатів, об'єднує існуючі впорядковані сегменти з існуючих рішень, в той час як мутація випадковим чином змінює частини існуючого рішення на нові. Цей процес повторюється до тих пір, поки не буде знайдено оптимальне рішення [12].

При розв’язанні задачі з використанням ГА замість того, щоб кодувати конкретне рішення, формуються правила, яким рішення повинно відповідати, щоб бути прийнятим (що дозволяє вирішувати проблеми з великою кількістю можливих потенційних рішень) [11].

Для кожного підходу до машинного навчання існує один або кілька алгоритмів ШІ, що реалізують відповідний підхід до навчання. Як було вказано раніше, контрольоване навчання використовується для вирішення задач класифікації (для цільових змінних з дискретними значеннями) або завдань регресії (для цільових змінних з безперервними значеннями) [13]. Прикладами алгоритмів ШІ, які використовують цей підхід, є наївний байесовский алгоритм, дерева рішень, опорні векторні машини (SVM), рекурентні нейронні мережі (RNN) або згорткові нейронні мережі (CNN). RNN зазвичай застосовуються в послідовних даних, а CNN є звичайним рішенням для великомасштабних вхідних даних [15], таких як послідовні дані або зображення.

Для навчання без вчителя зазвичай використовуються алгоритми k-середніх (де користувач вказує бажану кількість кластерів) або ієрархічну кластеризацію (де користувачеві не потрібно фіксувати певну кількість кластерів). Обидва ці методи намагаються групувати об'єкти разом на основі їх подібності [11].

Автоенкодери також можуть використовуватися для отримання ознак, створюючи нові ознаки з вихідного набору ознак за допомогою ШНМ. Автоенкодери можуть використовуватися для вирішення безлічі завдань, таких як: виявлення аномалій, шумозаглушення, попереднє навчання, виявлення подібності або створення вихідних даних вручну. Автоенкодери зазвичай є неконтрольованим засобом досягнення контрольованої мети [14].

1.3.3 Байєсовський алгоритм. Ще один метод ШІ – це байєсовській підхід, який також є неконтрольованим алгоритмом [11], але на відміну від рішень машинного

навчання (які можуть тільки визначити кореляцію в наборах даних), його можна використовувати для виявлення причинно-наслідкових зв'язків між функціями. На відміну від звичайного машинного навчання з підкріпленням, застосування байєсовського підходу полягає в тому, щоб досягти мети, а не дивитися на набори даних, за якими здійснюється навчання. Саме середовище може бути відомою або невідомою величиною залежно від використовуваного алгоритму (алгоритми на основі існуючих моделей або без моделей відповідно). Прикладами часто використовуваних алгоритмів є мережі Q-Learning та Deep Q-Learning Networks (DQN) [12].

1.4 Застосування методів ШІ в методологіях DevOps і Agile

Однією з основних перешкод на шляху використання методів ШІ в тестуванні ПЗ є відповідь на питання – які вимоги повинно висувати використання ШІ до процесу розробки ПЗ та до команди розробників? Методології DevOps і Agile сприяють автоматизації тестування в максимально можливій мірі. Разом з тим слід зазначити, що більшість технологій ШІ та МН залишаються невідомими для більшої частини спільноти розробників програмного забезпечення.

Методологія Agile – це модель розробки програмного забезпечення, заснована на Agile Manifesto, яку було створено на основі документування процесу розробки ПЗ. У той час, коли Agile безпосередньо застосовується для розробки ПЗ, DevOps діє як міст між його розробкою та експлуатацією.

Метод гнучкої розробки ПЗ Agile заснований на дванадцяти принципах. Серед принципів, які можуть отримати значне покращення від використання методів ШІ:

Задоволеність клієнтів та безперервна підтримка – необхідність безперервного тестування, яке може бути забезпечено за рахунок автономності ШІ;

Вітаємо мінливі вимоги – адаптивність ШІ дозволяє змінювати тести у відповідності до змін вимог та функцій ПЗ;

Працездатний продукт необхідно випускати якомога частіше, від кількох тижнів до кількох місяців, з перевагою більш коротких термінів – використання автоматичного тестування скорочує час, необхідний для тестування. Адаптивність ШІ, з іншого боку, скорочує час на підтримку автоматизованих тестів.

Таким чином, застосування методів ШІ в тестуванні ПЗ допомагають команді розробників винайти способи прискорення виконання робіт та скорочення їх обсягу [16]. Це дозволяє зробити висновки, що все, що можна автоматизувати, має бути автоматизовано таким чином, щоб людський досвід міг бути зосереджений саме там, де автоматизація не може бути використана.

1.5 Постановка задачі дослідження

Метою роботи є дослідження ефективності застосування методів ШІ для функціонального тестування програмного забезпечення. Проведений в рамках дослідження аналіз першоджерел підтвердив відсутність єдиного підходу до оцінки ефективності застосування методів ШІ для тестування додатків та чітко визначених шляхів їх вдосконалення. Серед якісних критеріїв ефективності дослідники у своїй більшості звертали увагу на виключення «людського фактору» з процесу тестування.

Тестування ПЗ проводиться з метою виявлення ненавмисних дефектів, і тест визнається успішним, якщо вдається успішно виправити ці дефекти. Тестування також має на меті показати відповідність програмних функцій системи вимогам, що визначені специфікаціями [1].

Проведення тестування методом «чорного ящика» спрямовано на виявлення дефектів в декількох категоріях – неправильні або відсутні функції системи, недоліки інтерфейсу, дефекти в структурах даних або під час доступу до зовнішніх баз даних, дефекти продуктивності, дефекти ініціалізації та завершення тощо.

Відсутність чітко визначеної передбаченої поведінки моделі системи, формально встановленого рівня відповідності між функціями системи та її

специфікацією є джерелом додаткових незручностей під час інтерпретації результатів тестування.

Пошук серед різних методів та технологій ШІ, пов'язаних з тестуванням програмного забезпечення, який проводився як серед академічних публікацій, так і в онлайн-матеріалах розробників інструментальних засобів тестування ПЗ, матеріалах конференцій і тренінгів, що проводились фахівцями з тестування, не виявив чіткої стратегії їх використання. Базуючись на проведеному аналізі наукової літератури можна дійти висновку, що більшість дослідників під час проведення тестування ПЗ методами ШІ використовували саме ГА – вони формували тестові випадки на основі вхідних даних, які відображались у вихідних даних. Застосування ГА для формування колекцій тестових випадків дозволило отримати досить високу ймовірність виявлення помилки.

Ще один метод ШІ, який знайшов своє використання під час тестування ПЗ, базується на визначенні відповідності між функціями програмної системи та специфікаціями за допомогою «оракула», який дозволяє визначити, чи пройдено системою тест чи ні.

Метою атестаційної роботи є дослідження використання методів ШІ для проведення тестування програмного забезпечення з оцінюванням результатів тестування. Проведення процедури тестування буде здійснюватися у два етапи – проектування досліджень та аналіз результатів. Для досягнення мети дослідження в роботі планується створить модель ШНМ для прогнозування рівня точності отриманих результатів при тестуванні ПЗ методом «чорного ящика». Для навчання ШНМ, оцінювання її адекватності, тестовий випадок буде перевірений шляхом тестування програмного забезпечення у відповідності до класів еквівалентностей – позитивного та негативного набору даних, після чого цей набір даних буде використовуватися під час навчання ШНМ та частково буде використаний у якості вхідних даних при перевірці роботи нейронної мережі.

Узагальнену схему проведення дослідження зображено на рис. 1.6.

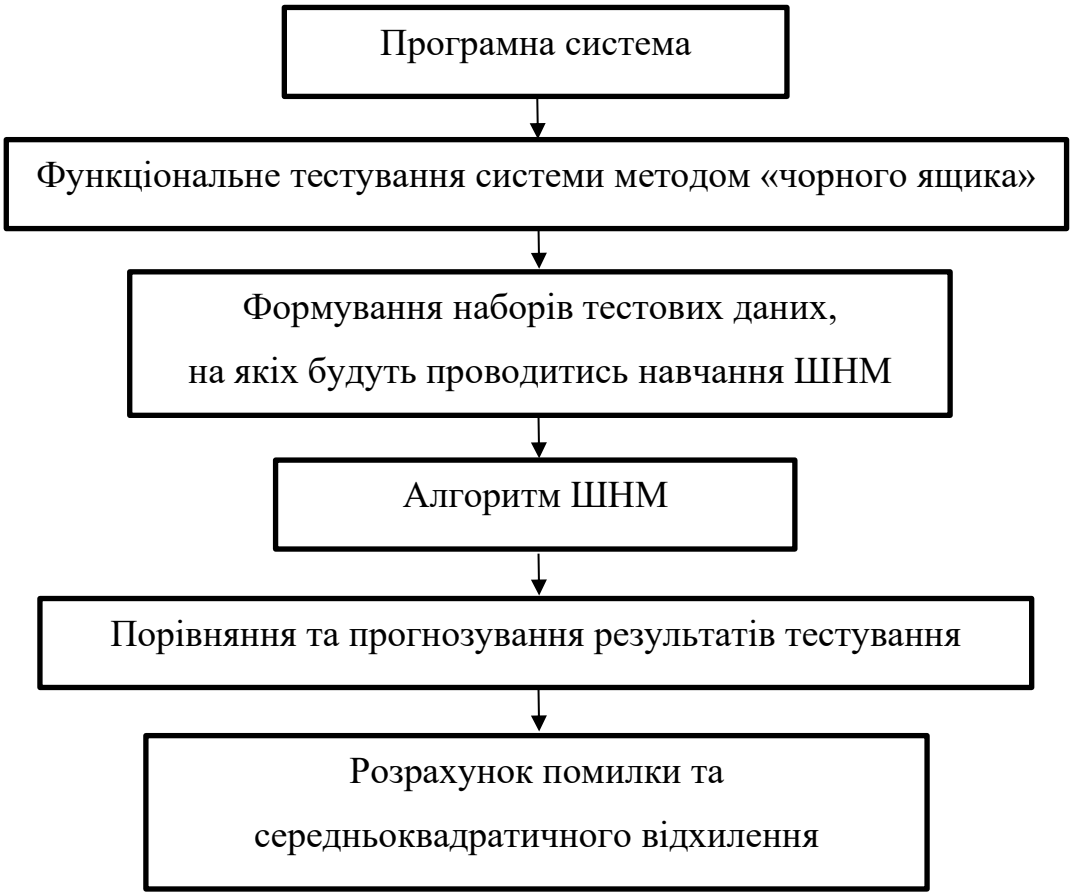

Рис. 1.6 – Алгоритм використання ШНМ в тестуванні ПЗ

Для досягнення мети дослідження виконаємо деталізацію локальних задач, які необхідно розв’язати в роботі.

Локальна задача 1. Необхідно відповідно до специфікації визначити набори тестових даних.

Локальна задача 2. Поділити отримані набори у відповідності до класів еквівалентностей.

Локальна задача 3. Визначення наборів даних, що відносяться до навчальної виборки та до тестового набору.

Локальна задача 4. Встановити та нормалізувати вагові коефіцієнти для кожного з наборів даних.

Локальна задача 5. Розробити алгоритм динамічної зміни вагових коефіцієнтів з використанням ШНМ для досягнення мети дослідження.

2 ЗАСТОСУВАННЯ МЕТОДІВ ШІ ДЛЯ ТЕСТУВАННЯ ПЗ

2.1 Огляд опублікованих результатів наукових досліджень

Пошук методів використання машинного навчання, пов'язаних з тестуванням програмного забезпечення, проводився як серед академічних публікацій та матеріалів наукових конференцій, так і в матеріалах розробників інструментальних засобів тестування ПЗ. З метою отримання інформації про ШІ та його використання для тестування програмного забезпечення були проаналізовані окремі наукові пошукові системи та сховища, серед яких були Google Scholar (https://scholar.google.com), Scopus (https://www.scopus.com) та Web of Science (https://apps.webofknowledge.com). Пошукові запити виконувались за ключовими словами «artificial-intelligence» (штучний інтелект), «software-testing» (тестування програмного забезпечення) та з обома ключовими словами.

Проведені дослідження показують, що ключові слова «штучний інтелект» та «тестування програмного забезпечення» набули експоненціальної популярності за останні кілька десятиліть. Результати пошукових запитів показують, що протягом останніх трьох років ШІ в основному застосовувався для тестування методом «чорного ящика». Отримані результати від різних пошукових систем свідчать, що в Google Scholar протягом останніх 3 років найпопулярнішим методом МН, який застосовувався при тестуванні програмного забезпечення, було контрольоване навчання, за яким слідує навчання з підкріпленням. На відміну від цих результатів, для Scopus Elsevier та Web of Science навчання з підкріплення було найпопулярнішим.

Хоча і не настільки популярні, як два згадані вище методи, безконтрольне навчання виявляється дуже поширеним явищем при тестуванні методом «чорного ящика». Тестування методами білого та сірого ящиків майже не використовують методи ШІ.

Найбільш популярними алгоритмами ШІ є «кластеризація» (загальний метод неконтрольованого навчання), ШНМ (використовується в навчанні під контролем та підкріплення) та ГА (зазвичай використовується в навчанні з підкріпленням). В

роботах розглядаються алгоритми ШНМ, що використовують навчання з підкріпленням під час проведення регресійного тестування [17], алгоритми оптимізації, що використовуються для зменшення розміру тест-кейсів [18, 19], та «нечіткого» тестування, яке використовується для дослідження шляхів керування системою [20].

На підставі проведеного аналізу можна зазначити, що метод тестування «чорного ящика» найбільше виграє від застосування методів ШІ. Всі три методи МН (контрольований, неконтрольований та з підтвердженням) досить широко використовуються. Найбільш поширеними серед алгоритмів ШІ, що використовуються для тестування програмного забезпечення, є кластеризація, ШНМ та ГА, які знаходять своє застосування при проведенні нечіткого та регресійного тестування. Результати проведеного аналізу доводять, що більшість варіантів використання методів ШІ використовується як форма оптимізації вже існуючих методів. Алгоритми ШІ у більшості випадків орієнтовані на охоплення якомога більшу частину коду з найменшою кількістю тестових випадків. ШІ застосовують для:

– пошуку ефективних способів формування тестових наборів даних, що дозволяють охопити увесь код;

– розстановки пріоритетів для тестових випадків;

– зменшення кількість тест-кейсів за рахунок фільтрації тестів з більшим охопленням, менш трудомістких або з більшою історією находження багів.

В роботі [21] вказано: «…ключові переваги тестування, керованого штучним інтелектом, полягають в тому, що воно є універсальним, багаторазовим, надійним, адаптивним та гарно масштабується». Підходи машинного навчання можна використовувати для тестування різних рівнів тестування, атрибутів якості і додатків в різних доменах. Одні і ті ж тести можуть використовуватися в декількох додатках в межах домену або між доменами завдяки їх незалежності. Наприклад, за допомогою методів навчання з підкріпленням можна постійно створювати нові тести, що дозволяє запобігти парадоксу пестицидів, який може стати проблемою при традиційній автоматизації тестування.

Парадокс пестицидів виникає, коли одні і ті ж тести виконуються багаторазово, і автоматизоване тестування перестає визначати нові помилки. Комбінуючи створення тестів на основі ШІ і запуск декількох агентів тестування з хмари можна значно покращити якість тестуванням [22].

2.2 Використання алгоритмів ШНМ в циклі розробки ПЗ

Існує досить велика кількість організації та асоціацій з підтримки розробки ШІ та машинного навчання та таких, що займаються вдосконаленням методів тестування програмного забезпечення, але при проведенні дослідження предметної області була виявлена тільки одна асоціація, що поєднує ці дві практики – Асоціація штучного інтелекту для тестування програмного забезпечення (AISTA). Щоб відповісти на питання, яке місце займає ШІ в тестуванні програмного забезпечення, AISTA дала таке визначення: «Це нова область, що спрямована на розробку систем штучного інтелекту для тестування програмного забезпечення, методів тестування систем зі штучним інтелектом та, у підсумку – на розробку програмного забезпечення, здатного до самоперевірки і самозцілення» [23].

ШІ створює шар абстракції поверх програмного забезпечення, яке підлягає тестуванню, так саме, як людський мозок створює абстракцію програмного забезпечення та його очікуваної поведінки. «Людський мозок має уявлення і розуміння, що, наприклад, в додатку інтернет-магазину є товари для покупок із зображеннями і описами, поля пошуку, кошик для покупок, функція входу в систему та інші функції. Ми хочемо, щоб машина думала і діяла як людина, і в досягненні цієї мети ШІ є доброю підмогою» [24].

Для вирішення зазначених завдань може бути використаний метод, який базується на використанні ШНМ, оскільки, в загальному випадку, програмне забезпечення може бути розглянуто як «чорний ящик» в багатовимірному просторі

вхідних даних. Така модель може досить ефективно використовувати методи нейронних мереж.

Мозок людини є надзвичайно складною, нелінійною, паралельною системою обробки інформації, що має здатність організовувати свої структурні компоненти. Для того, щоб досягти високої продуктивності, нейронні мережі використовують безліч взаємозв'язків між елементарними осередками обчислень – нейронами. Аналогічним чином в ШНМ визначається взаємодія між штучними нейронами.

Модель нейрона, що лежить в основі ШНМ, представлено на рис. 2.1.

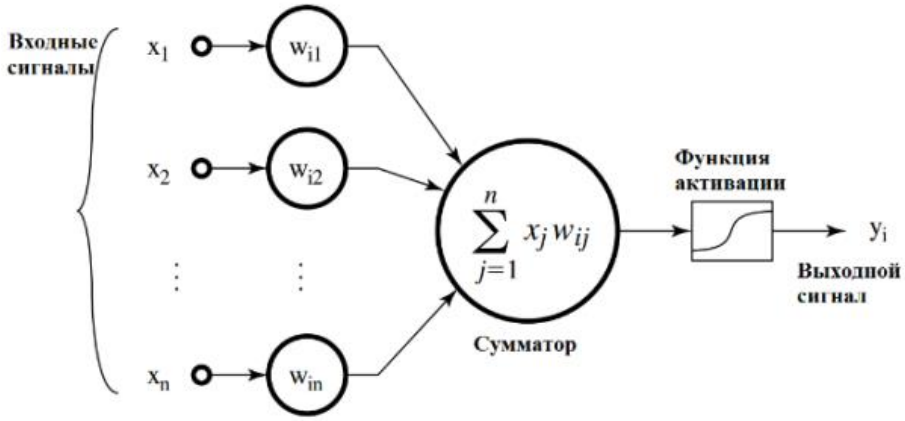

Рис. 2.1 – Модель штучного нейрону

Штучні нейронні мережі являють собою систему з'єднаних і взаємодіючих між собою простих процесорів (штучних нейронів), кожен з яких має справу з сигналами, які він періодично отримує, і сигналами, які він періодично посилає іншим процесорам. Штучний нейрон імітує в першому наближенні властивості біологічного нейрона: на вхід штучного нейрона надходить деяка множина сигналів, кожен з яких є виходом іншого нейрона. Кожен вхід множиться на відповідну вагу, що аналогічна синаптичній силі, і всі твори підсумовуються, визначаючи рівень активації нейрона [11].

Тоді в математичному поданні функціонування нейрона можна описати системою рівнянь:

$$net = \sum_{j=1}^{n} x_j w_{ij} \quad (3)$$

$$y_i = \frac{1}{1 + e^{-net}} \quad (4)$$

де x_1 , x_2 ,..., x_n – вхідні сигнали; w_{i1} , w_{i2} ,...,w_{in} – вагові коефіцієнти нейрону i; y_i – вихідний сигнал нейрону.

Нейрони за своїми властивостями можна поділити на три групи: вхідні нейрони, вихідні нейрони і нейрони прихованого шару (обчислювальні вузли). Багаторівнева ШНМ прямого поширення складається з вхідного шару (один вузол для кожного вхідного значення), одного або декількох прихованих шарів і вихідного рівня. В якості методу навчання нейронної мережі прямого поширення, як правило, використовується метод зворотного поширення помилки.

Алгоритм полягає в передачі вхідного сигналу в прямому напрямку і сигналу помилки – в зворотному. При прямому проході вектор вхідних сигналів подається на вхідний шар ШНМ, після цього він поступово поширюється по мережі від одного шару до наступного. В результаті утворюється набір вихідних сигналів, які представляють собою фактичну реакцію мережі на вхідний набір.

При прямому проході всі синаптичні ваги постійні, а під час зворотного проходу ваги налаштовуються відповідно до правила корекції помилок. Налаштування ваг відбувається наступним чином: фактичний вихід мережі віднімається з очікуваного і формується сигнал помилки. Цей сигнал поширюється мережею в напрямку, протилежному напрямку синаптичних зв'язків. Описану структуру мережі наведено на рис. 2.2 [12].

Фактично нейронна мережа – це особливий спосіб завдання функції. Для того, щоб працювати з нею, необхідно, в першу чергу, навчити мережу.

Процес навчання полягає у підготовці навчальної вибірки, що представляє собою множину пар вхідних та відповідних їм свідомо вірних вихідних векторів. Елементи такої множини повинні бути незалежними для можливості розширення вибірки та подання її в довільному порядку. Навчальна вибірка проходить через

ШНМ, синаптичні ваги в нейронних зв'язках коригуються таким чином, щоб визначити функцію, що задовольняє навчальній вибірці. При цьому якість навчання визначає якість роботи ШНМ.

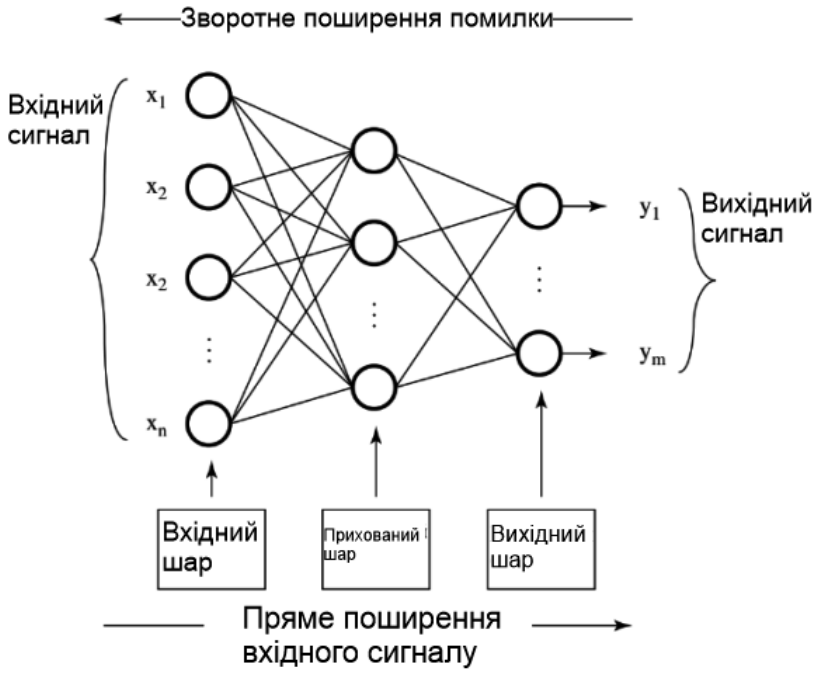

Рис. 2.2 – Структура багатошарової нейронної мережі

Другим етапом є перевірка результатів навчання ШНМ. З цією метою готується тестова вибірка, аналогічна навчальній, але з використанням інших значень набору. Якщо результат тестування буде визнаний незадовільним, необхідно провести донавчання мережі або змінити її структуру.

Багаторівнева нейронна мережа може бути навчена відповідно до тестування додатку на основі згенерованих в довільному порядку вхідних даних, які відповідають технічним завданням або специфікації. В результаті навчання ШНМ здатна показувати достатній ступінь точності для програмного забезпечення, що тестується. Таким чином, навчена ШНМ стає симульованою моделлю програмного продукту [16].

Згодом, коли буде доопрацьовано функціонал розроблюваного програмного продукту і створені нові версії програми, стає необхідним проведення регресійного тестування. Цей вид тестування спрямований на перевірку змін, реалізованих в додатку або навколишньому середовищу. В якості таких змін може виступати новий

функціонал, виправлення дефекту, міграція на нову базу даних, операційну систему або сервер додатків тощо. В такому випадку регресійне тестування проводиться з метою перевірки працездатності вже існуючої функціональності.

Таким чином, в разі застосування в процесі тестування ШНМ при регресійному тестуванні програмний продукт перевіряється на тестових даних для отримання вихідних результатів, які потім порівнюються з результатами роботи ШНМ [9].

2.3 Методика застосування ШНМ в тестуванні ПЗ

Припустивши, що після внесення виправлень в програмний продукт існуючий функціонал не зміниться, можна очікувати, що при однакових вхідних даних нейронна мережа і реальний програмний додаток дадуть ідентичні результати. За допомогою порівняння можна буде зробити висновок, чи є коректним результат тестування програми [16]. Вказану методику тестування представлено на рис. 2.3.

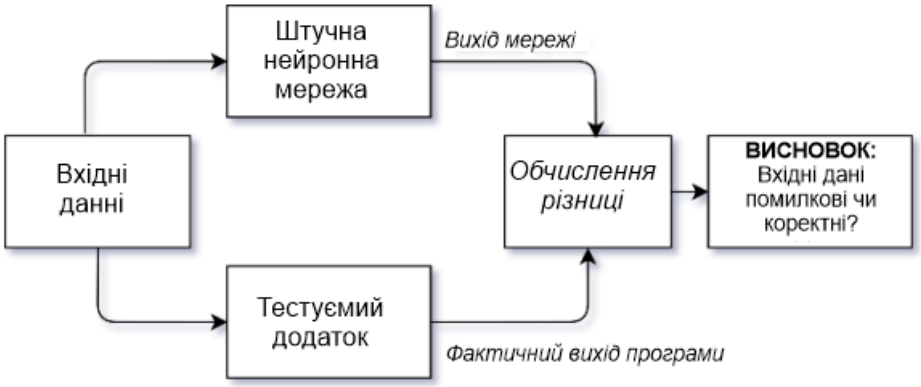

Рис. 2.3 – Методика тестування з використанням ШНМ

Використання моделі ШНМ починається з етапу навчання. На цьому етапі (рис. 2.4) вектор введення (тестовий приклад) для кожного тест-кейсу генерується випадковим чином, відповідно до специфікації програми. Кожен вхідний вектор подається на вхід програми, яка генерує відповідний вихідний вектор.

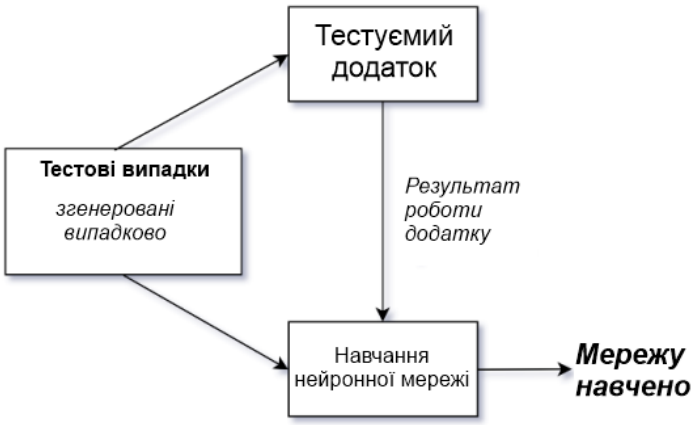

Рис. 2.4 – Фаза навчання ШНМ

Вхідні та вихідні вектори програми використовуються для навчання нейронної мережі. Навчена нейронна мережа може виконувати функцію автоматизованого «оракула» для тестування наступних версій програми в процесі регресійного тестування [25].

На етапі тестування (рис. 2.5) кожен тестовий випадок подається як вхідний вектор одночасно і на вхід програмної системи, що підлягає тестуванню, і на вхід навченої ШНМ. Для кожного вхідного вектору інструмент порівняння обчислює абсолютну відстань між виходом ШНМ та відповідним значенням виходу програми.

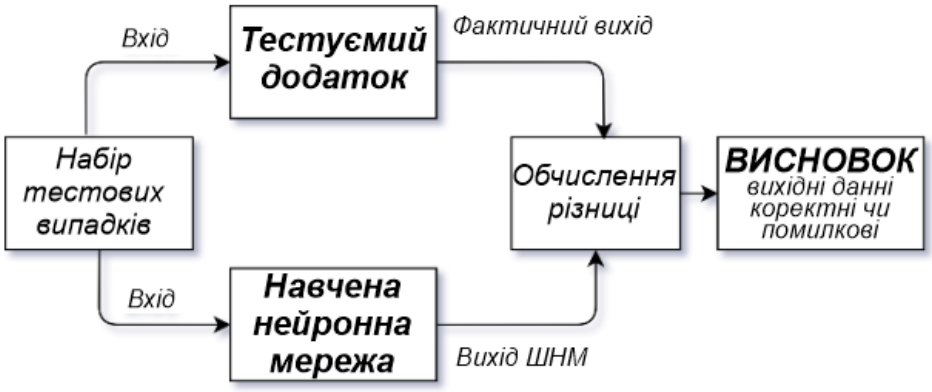

Рис. 2.5 – Фаза тестування

Всі виходи є числами від нуля до одиниці. Обчислена відстань потрапляє в один із трьох інтервалів, визначених двома пороговими значеннями в діапазоні [0, 1]. Якщо передбачення мережі та вихід програми однакові, а обчислена відстань потрапляє в

інтервал між 0 і низьким порогом відмінності (відстань дуже мала), то це означає, що передбачення мережі відповідає виходу програми, і обидва результати, ймовірно, будуть правильними . У цьому випадку інструмент порівняння повідомляє, що вихід програми правильний. З іншого боку, якщо результат прогнозування мережі відрізняється від виходу програми і відстань між виходами дуже велика (потрапляє в інтервал між високим порогом і 1), то інструмент порівняння повідомляє про помилку програми (неправильний вихід). В обох випадках висновок ШНМ буде правильним, і він є достатньо надійним для оцінки правильності вихідних даних програми. Однак, якщо відстань розміщується в інтервалі між низьким і високим порогом, вихід мережі вважається ненадійним. В цьому випадку засіб порівняння повідомляє про помилку програми лише в тому випадку, якщо передбачення мережі повністю ідентичне виходу програми.

Інструмент порівняння використовується як незалежний метод порівняння результатів моделювання ШНМ та результатів тестування ПЗ. Необхідно забезпечити об’єктивний автоматизований підхід, який би гарантував, що на результати роботи програми не впливають зовнішні фактори. Це фактично замінює тестера, який може бути упередженим, маючи попередні знання про функціональні можливості програми.

Інструмент використовує вихід нейронної мережі та результат тестованої програми. Відстань між виходами обчислюється як абсолютна різниця між значенням вузла ШНМ для кожного виходу та відповідним значенням у додатку. Оскільки для забезпечення мережевих виходів використовується функція активації, значення активації вихідних вузлів ШНМ може становити число від 0,0 до 1,0. Відповідне значення виходу програми дорівнює 1,0, якщо прогнозований та фактичний результати співпадають. В іншому випадку воно дорівнює 0,0. Таким чином, відстань охоплює діапазон між 0,0 і 1,0, і ми використовуємо це значення, щоб визначити, чи правильно спрацювала програма, або ми отримали невірний результат її роботи [26]. В таблиці 2.1 наведено чотири можливих стани виходів програми та ШНМ.

Табл. 2.1 – Виходи програмної системи та ШНМ

		Негативний результат роботи програми	
		Вірно	Невірно
Вихід ШНМ	Вірно	Категорія 1 Дійсно позитивний	Категорія 2 Дійсно негативний
	Невірно	Категорія 3 Ложно негативний	Категорія 4 Ложно позитивний

Оскільки ШНМ є лише моделлю реальної програмної системи, деякі її результати можуть бути неправильними. З іншого боку, сама програма, що тестується, може містити дефекти, виявлення яких і є основною причиною процесу тестування. Якщо вихід ШНМ вважати правильним, а вихід програми – невірним, то оцінка інструменту порівняння класифікується як дійсно негативний результат – визначення того факту, що результат роботи програми містить фактичну помилку. Аналогічним чином інші три категорії класифікації надають можливість класифікувати вихідних даних. Кожен вихід ШНМ та програми, яка тестується, оцінюється таким чином. Хоча найбільший інтерес з точки зору тестера становить пошук неправильних результатів (категорії 2 та 3), слід зазначити, що немає видимої різниці, коли вихід ШНМ є таким самим, як і результат випробуваної програми (категорії 1 і 3). Категорії 2 і 4 також подібні в цьому відношенні, оскільки або вихід мережі є правильним, або результат роботи програми – правильний, причому перший з наведених варіантів є більш імовірним. ШНМ навчають моделювати оригінальну програму, однак вона не здатна класифікувати вихідні дані на сто відсотків правильно через проблему збіжності помилок. Таким чином, найбільший інтерес представляють випадки, коли результат роботи програми на обраному наборі вхідних даних тестування є неправильними: категорії 2 та 3 у позначеннях табл. 2.1.

Отже, категорії 1 і 3 повинні відрізнятися одна від одної під час застосування інструменту порівняння; подібне розділення також необхідно виконувати і для категорій 2 і 4.

Інструмент порівняння по-різному обробляє два типи виходів, оскільки існує чотири можливі ситуації для двійкового виводу та п’ять для безперервного виводу. Коли і вихід ШНМ, і результат неправильного функціонування програми різні, може спостерігатися ситуація, коли і вихід ШНМ може бути правильним, і вихід неправильно функціонуючого додатку також може бути правильним. Також може виникати ситуація, коли обидва виходи (і ШНМ, і програмного додатку, що містить дефект) можуть бути помилковими, тоді як для двійкового стану системи можливе лише бінарне значення виходу (див. табл. 2.2).

Табл. 2.2 – Можливі типи негативних виходів

		Результат порівняння виходів між собою	
		Однакові	Різні
Тип виходу	Дискретний	1) Обидва вірні 2) Обидва невірні	1) ШНМ вірний 2) Програма з дефектом - вірний
	Безперервний	1) Обидва вірні 2) Обидва невірні	1) ШНМ вірний 2) Програма з дефектом – вірний 3) Обидва невірні

Дизайн експерименту розділений на три частини, одну для прикладу, одну для нейронної мережі, а іншу для інструменту порівняння, який обчислює відстань між двома виходами (див. рис. 2.5). Специфікація та алгоритм заявки на затвердження кредитної картки використовуються для забезпечення необхідного формату вхідних та вихідних атрибутів та розміщення введених несправностей. Дизайн нейронної мережі також частково залежить від формату вводу/виводу програми; однак налаштування для навчання нейронної мережі визначались вручну методом спроб і помилок. На етапі навчання вихідні дані для тренування нейронної мережі та перевіреної програми генеруються випадковим чином. Вхідні дані подаються в тестовану програму в якості вхідного вектору, який генерує вихідний вектор для кожного навчального прикладу. Таким чином, набір вхідних векторів та вихідних векторів передаються як вхідні дані до дворівневої нейронної мережі, яка навчається

за допомогою алгоритму зворотного розповсюдження. На етапі оцінки створюється мутована версія тестованої програми шляхом внесення по черзі змін до початкової програми. Далі тестування передається як у нейронну мережу, так і в тестовану програму [16].

На наступному етапі результати як перевіреної програми, так і навченої мережі обробляються засобом порівняння, який приймає рішення про помилковий чи правильний результат випробуваної програми. Процес, за яким слідує експеримент, починається з генерації тестових кейсів. Вхідні атрибути створюються за допомогою специфікації програми, яка тестується, тоді як вихідні дані генеруються виконанням перевіреної програми. Дані проходять процедуру попередньої обробки, в якій усі безперервні вхідні атрибути нормалізуються (діапазон визначається шляхом знаходження максимальних і мінімальних значень для кожного атрибута), а двійковим входам і виводу призначаються значення 0 або 1. Безперервний вихід обробляється по-різному, і вихід кожного прикладу розміщується у правильному інтервалі, заданому діапазоном можливих значень та кількістю використовуваних інтервалів. Оброблені дані використовуються як набір даних для навчання нейронної мережі. Параметри мережі можуть визначатися перед початком навчання алгоритму ШНМ. Навчання мережі включає подання усього набору даних за один інтервал тестування, а також – кількість інтервалів для навчання. Алгоритм навчання зворотного розповсюдження робить висновок, коли досягнуто максимальної кількості інтервалів навчання або досягнуто мінімальної частоти помилок. Потім мережа може бути використана у якості оракулу для прогнозування правильних результатів при побудові наступних тестів.

У дослідженні [10] тестування систем з використанням МН було розділене на дві наукових спільноти: співтовариство машинного навчання і співтовариство тестування програмного забезпечення. Дослідники підкреслюють, що ці два спільноти по-різному інтерпретують сам термін «тестування». Спільнота машинного навчання фокусується на вдосконаленні моделі з використанням МН, щоб оцінити її точність і підвищити ефективність передбачення. Тестування відбувається в процесі навчання моделі МН відповідно до передової практики спільноти з розділення набору

даних на набори даних для навчання і для тестування, а в деяких випадках – для перевірки. Для спільноти тестування програмного забезпечення тестування фокусується на будь-яких вхідних даних, навіть якщо очікуваний результат невідомий або не визначений. При тестуванні на рівні інтеграції або системи компонент МН тестується у взаємодії з іншими компонентами системи або як система в цілому. При цьому слід враховувати всі атрибути якості програмного забезпечення, до яких відносяться правильність і надійність [27].

Те, яким чином два співтовариства інтерпретують тестування, вимагає стандартизації термінології тестування. Команда тестувальників в недалекому майбутньому буде складатися з фахівців по роботі з великими даними, архітекторів машинного навчання, тестувальників програмного забезпечення, бізнес-фахівців. В дослідженні [27] виділяють три фундаментальні проблеми: відсутність механізму розробки та використання тестових оракулів; великий простір вхідних даних, що з легкістю призводить до комбінаторного вибуху; та відсутність стандартизованих методів впровадження в алгоритми МН таких метрик якості, як справедливість та етичність. Джонсон підкреслює в своїй статті «необхідність в методологіях, які забезпечують більшу довіру, прозорість і контроль за справедливістю рішень, що приймаються за допомогою ШІ – це позитивна мета, що досягається за рахунок усунення негативних наслідків упередженості та інших форм нерівності» [28].

3 ОПИС МЕТОДУ ДОСЛІДЖЕННЯ

3.1 Модель тестування ПЗ

Модель тестування програмного забезпечення було запропоновано Полом Джерардом (Paul Gerrard) у 2017 р. та неодноразово обговорювалась на різних семінарах та презентаціях, в тому числі – на конференції FiSTB, яка відбулась у вересні 2019 року. На думку Джерарда, старі методи проведення тестування не будуть працювати у майбутньому у зв'язку з відсутності у них гнучкості в налаштуваннях, тиску на тестувальників з боку розробників – цифровізація ускладнює системи і відповідальність за тестування перерозподіляється на всю команду розробників [29].

Описану Джерардом узагальнену модель тестування ПЗ наведено на рис.3.1.

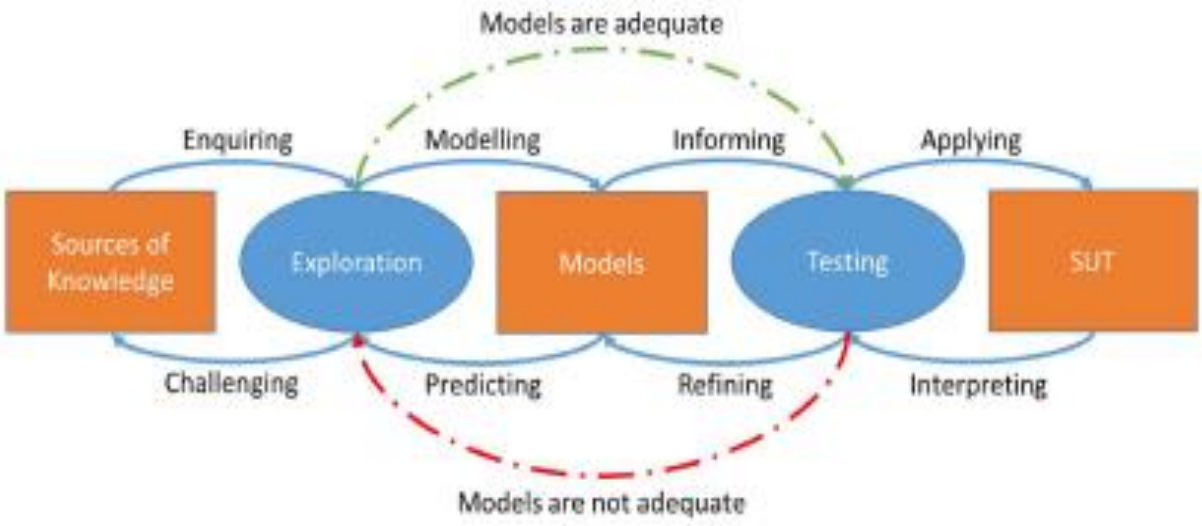

Рис. 3.1 - Нова модель тестування (Джеррард, 2017 г.)

Запропонована модель описує процес тестування ПЗ як спосіб мислення тестувальника. Цей процес можна повністю узагальнити, щоб не залежати від логістики тестування – не має значення, які інструменти або процеси використовуються або дотримуються для досягнення мети тестування [29]. Модель повністю відповідає процесам тестування та підходить для використання у якості еталонної моделі для організації процесу тестування з використанням методів ШІ.

Тестування з використанням алгоритмів ШІ, як вже було вказано вище, засновано на використанні інтелектуальних самонавчаємих агентів, які можуть автономно сприймати навколишнє середовище і діяти в ньому. Вони можуть планувати і створювати тестові приклади в цільовій системі, досліджуючи функціональні можливості і дізнаючись про використання додатку. Зрештою, вони можуть самостійно виконувати тести і оцінювати отримані результати тестування. Агенти, що організовані з іншими агентами, можуть діяти на різних ієрархічних рівнях [21].

Описаний вище підхід до тестування ПЗ на підставі алгоритмів ШІ передбачає використання одного або декілька агентів тестування, здатних досліджувати і тестувати цільовий додаток. Агенти можуть бути використані у двох ролях:

– агент-координатор;

– тестовий агент.

Агенти-координатори базуються на використанні циклу управління MAPE-K, представленого фірмою IBM в офіційному документі «An architectural blueprint for autonomic computing» [30]. MAPE – це скорочення від слів «Моніторинг, Аналіз, Планування та Виконання», що базуються на знаннях, які можуть бути отримані за допомогою машинного навчання.

Функціональне середовище тестової системи представлено на рис.3.2.

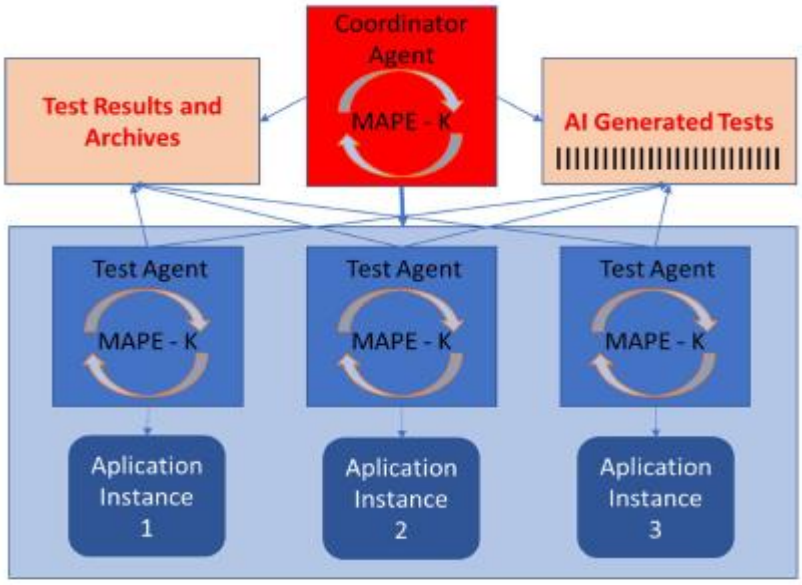

Рис.3.2 – Функціональне середовище агентів тестування

Відповідно до представленої на рис.3.7 схеми взаємодії, тестові агенти автономні та повинні діяти в середовищі цільового додатку незалежно один від одного, не взаємодіючі між собою. Цього можна досягти за рахунок встановлення агентам відокремлених задачі тестування різних функції програми або використовувати контейнерну технологію (наприклад, докер для цільових програм), в результаті чого кожен з агентів буде працювати зі своїм власним примірником моделі подання бажаної поведінки системи, яка тестується (System Under Test, SUT).

На етапі аналізу і проектування тестів агенти тестування досліджують додаток, щоб отримати уявлення про ПЗ та середовище виконання. Вони несуть відповідальність за побудову моделі програми шляхом вивчення цільової системи. В рамках фази реалізації агенти створюють тестові набори для їх подальшого використання в наборах тестів моделей програмної системи.

На етапі виконання агенти виконують тести і зберігають результат в архівах результатів тестування. Вони взаємодіють з ПЗ, яке тестується, вводять дані та порівнюють отримані результати. Одночасно вони навчаються на результатах взаємодії і постійно оновлюють моделі додатків.

3.2 Тестове покриття

Застосування методів ШІ повинно докорінно змінити процес тестування програмного забезпечення. Тестувальники роблять швидкі висновки на основі аналізу даних. ШІ робить ще більш швидкі судження на основі набагато більшої кількості даних. ШІ не залишить без змін жодну область тестування програмного забезпечення – стверджує Arbon [31], описуючи, як ШІ вплине на процес тестування ПЗ у майбутньому.

«Тестування програмного забезпечення – це динамічна перевірка програми з використанням кінцевих тестів у нескінченній області виконання» – у своєму визначенні тестування програмного забезпечення Тарік Кінг [22] підкреслює головну

проблему динамічного тестування. На його думку, перед тестером стоїть нездійсненне завдання – все перевірити. Складність програмного забезпечення збільшується експоненціально, коли в систему додаються нові функції, підпрограми та їх інтеграції. Звичайне тестове покриття при ручному тестуванні може збільшуватися тільки лінійно, коли кількість тестових ресурсів залишається незмінним.

Отже, пробіл в тестовому покритті збільшується. Саме сюди можуть прийти інструментальні засоби для підтримки проведення тестування з елементами ШІ та інтелектуальні тестові агенти для підтримки тестування [21]. Одним з таких інструментальних засобів є фазінг.

3.3 Інтелектуальний фазінг

Фазінг (Fuzzing) – це одна з технологій тестування програмного забезпечення, в якій замість очікуваних вхідних даних програмі передаються випадкові або спеціально сформовані дані [14]. Метою такого дослідження поведінки програмної системи є спроба знайти непередбачені розробниками ПЗ вхідні дані, які призводять до аварійного завершення програми або до її некоректної поведінки. Вхідні дані можуть передаватися через файли, мережеві сокети, API, змінні оточення та тощо. У тому випадку, якщо програма входить в нескінченний цикл або аварійно завершує роботу, це вважається фактом виявлення дефекту в програмі, який може призвести до виявлення певної уразливості.

Одним з методів підвищення якості фазінгу є використання в контурі управління методів ШІ. Тестування за допомогою нечіткого алгоритму – це автоматизований процес тестування програмного забезпечення шляхом надання програмі випадкових вхідних даних. Залежно від методики генерування вхідної інформації існуючі технології фазінгу можна умовно розділити на два класи:

1) мутація існуючих даних;

2) генерація нових даних.

Відповідно до методики формування вхідної інформації типи фазінгу також змінюються:

Тип 1. Заздалегідь підготовлені дані.

Тип 2. Використання випадкових даних. Є найменш ефективним підходом, оскільки причини виникнення відмови в роботі ПЗ в цьому випадку досить важко визначити.

Тип 3. Ручна зміна даних. Шляхом внесення завідомо помилкової вхідної інформації тестувальник штучно виклкає збій в роботи ПЗ. Ефективність такого підходу досить мала через відсутність автоматизації процесу.

Тип 4. Перебір мутацій даних. При цьому підході обсяг тестів зменшується, але, у той самий час, обсяг даних сильно збільшується, оскільки дані піддаються значним мутаціям.

Тип 5. Свідоме внесення змін у даних. Для проведення такого типу тестування виконується додатковий аналіз з метою визначення даних, які не будуть змінюватися, і даних, які підлягають змінам. Такий підхід вимагає значних витрат часу.

Спрощений алгоритм, який реалізує методику застосування фазеру для тестування ПЗ, наведено на рис. 3.3.

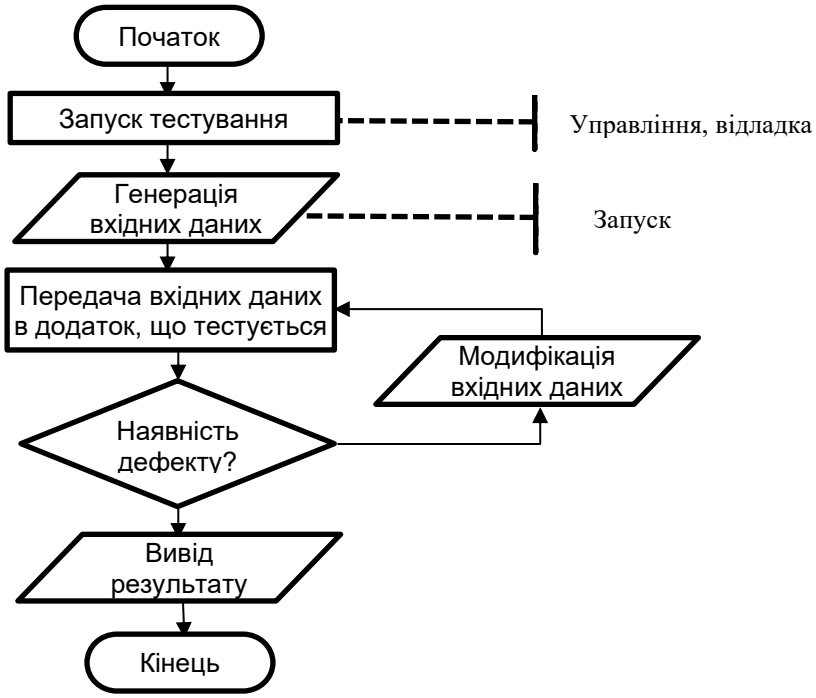

рис. 3.3 – Алгоритм фазінгу

Методика тестування ПЗ, що розробляється в рамках атестаційної роботи, включає в себе наявність так званого інтелектуального агента (intelligent agent) [6]. Програма, яка містить в своєму складі інтелектуального агента – це система, яка самостійно виконує вказані користувачем завдання протягом тривалих проміжків часу. В даному випадку термін «інтелектуальний» підкреслює більш високий рівень технології управління у порівнянні з примітивними (dumb) тригерними системами автоматичного тестування. Функції, що покладаються на інтелектуального агента, можуть бути наступними (рис.3.4):

– самостійне виконання завдань, вказаних користувачем;

– автоматичне розпізнавання типу тестової задачі та необхідних для її виконання інструментів;

– запуск необхідних завдань у визначений користувачем час.

Рис. 3.4 – Схема функціонування інтелектуального агента

В системі задіяні вищевказані функції за рахунок використання цілеспрямованого інтелектуального агента. Цілеспрямований агент буде зберігати інформацію про тих ситуаціях, які для нього є бажаними, що дасть йому спосіб вибрати серед багатьох шляхів проведення тестування ПЗ саме той, який призведе до мети тестування.

3.4 Навчання інтелектуального агента

Навчання з підкріпленням (НзП) – це область МН, яка має справу з агентом та його взаємодією з навколишнім середовищем. Інтелектуальний агент – це об'єкт, який дізнається про навколишнє середовище на підставі виконання деяких дії та спостерігаючи за їх результатом. Середовище НзП моделюється як Марковський процес прийняття рішень, який формулює прийняття рішень у вигляді випадкового процесу. У середовищі НзП метою агента є максимізувати сукупну винагороду доти, поки воно не досягне кінцевого стану.

Кінцевий стан – це унікальне стан в середовищі, після досягнення якого агенту не потрібно робити ніяких подальших дій. Агент вибирає дію для кожного стану для досягнення наступного стану і отримує відповідну винагороду за цю дію. Наприклад, в грі в шахи агент повинен вибрати фігуру, щоб зробити хід. У середовищі маневрування робота агент повинен або йти вправо, або вліво, або продовжувати рух вперед, щоб досягти цільового місцеположення. Відповідно до цього, в тестуванні ПЗ агент повинен робити такі зміни у вхідних даних, які призводять до переходу в один з виключних станів програмної системи, який не дозволяє продовжити її цільове використання.

Нагорода – це скалярна величина, яка служить стимулом для агента на шляху до визначеної мети. Позитивна винагорода означає, що в результаті виконаної дії агент наблизився до свого цільового стану. Так саме негативна нагорода може означати, що дії агента віддаляють систему від цільового стану. Ґрунтуючись на досвіді, який агент отримує від дослідження, агент вчиться обирати оптимальні дії для кожного стану, щоб максимізувати сукупну винагороду.

В атестаційній роботі пропонується застосування алгоритму навчання з підкріпленням для спрямованого фазінгу. В контексті тестування ПЗ в роботі розглядається модель фазеру як агента НзП, а програму, що підлягає тестуванню, – як середовище його існування. Кожен набір вхідних даних програми – це стан агенту, а набір мутацій на вході програми – це можливі дії агента в цьому стані. Таким чином,

область пошуку для агента – це набір всіх можливих входів в програму, а кінцевий або цільовий стан – це будь-який набір вхідних даних програми, який досягає цільового місця в програмі.

3.5 Методика проведення досліджень

Процес тестування ПЗ тепер може бути представлений як вибір агента у стані вхідних даних, що намагається знайти такий набір параметрів програми, який дозволяє їй досягти цільового стану шляхом постійної зміни поточного введення програми. Агент переходить з одного стану в інший, накопичуючи винагороди при кожному переході. Оскільки максимальна винагорода, яку отримує агент, обмежена, то агент вчиться уникати негативних винагород, тобто мутації, які відхиляють його від цільового стану. Винагороди є єдиним джерелом інформації, яку може вивчати агент, тому важливо вибрати функцію корисності винагороди для агента.

У навчанні з підкріпленням приймають участь дві компоненти: агент та оточуюче середовище.

Агент: Агент в контексті навчання з підкріпленням – це учень, якому поставлене завдання зрозуміти навколишнє середовище. Агент вирішує, яку дію вибрати в тій чи іншій ситуації. В аспекті тестування ПЗ агент – це фазер, який обирає набір вхідних даних для виконання наступного етапу тестування, причому спочатку агент не знає ні правил, за якими виконується тестування, ні того, як саме відбувається сам процес тестування.

Навколишнє середовище: В навчанні з підкріпленням середовище – це система, яку агент прагне зрозуміти. Воно складається з усього, що не є агентом. Функціонування навколишнього середовища агенту апріорі невідомо.

На рис. 3.5 наведено схему взаємодії між агентом та навколишнім середовищем.

Рис.3.5 – Алгоритм навчання з підкріпленням

Стан: Під станом розуміють подання поточної ситуації в оточуючому середовищі.

Нагорода: Після кожної взаємодії агент отримує числове значення від середовища. Це числове значення – винагорода – описує результат хорошої або поганої дії агента. Висока позитивне винагорода означає, що дія, виконана агентом, було правильною, а негативна винагорода означає, що дія призвела до віддалення від мети тестування. Мета агента – максимізувати сукупну винагороду, яку він отримує від середовища за певний період часу. Сигнал винагороди можна змоделювати по-різному для одного середовища.

Стратегія: Стратегія описує перехід агента у відповідний стан. Це відображення стану навколишнього середовища на дію, яка повинна бути виконана агентом в цьому стані. Агент намагається вивчити оптимальну стратегію, щоб обрати найкращу дію для кожного стану.

4 ПРОВЕДЕННЯ ЕКСПЕРИМЕНТІВ ТА АНАЛІЗ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

4.1 Підготовка вхідних даних

Використання методів ШІ в тестуванні ПЗ можна віднести до елементів автоматизації тестування програмного забезпечення. Воно використовує програмні засоби для виконання тестів і перевірки їх результатів, що сприяє значному скороченню часу, необхідного для проведення тестування [33].

Для перевірки правильності гіпотези шодо ефективності використання нейромережевого підходу в тестуванні ПЗ використаємо цей підхід для тестування відокремленої підсистеми ІАС Університет для формування академічної довідки.

Відповідно до Положення про організацію освітнього процесу в Харківському національному університеті радіоелектроніки, «рейтинг здобувачів вищої освіти з навчальної дисципліни вимірюється за 100-бальною шкалою з подальшим переведенням в оцінку за національною шкалою та шкалою ЄКТС» [34]. Переведення здійснюється у відповідності до таблиці (рис.4.1).

Оцінка з дисципліни	Оцінка за національною шкалою		Оцінка за шкалою ЄКТС
	екзамен	залік	
96-100	5 (відмінно)	Зараховано	A
90-95	5 (відмінно)		B
75-89	4 (добре)		C
66-74	3 (задовільно)		D
60-65	3 (задовільно)		E
35-59	2 (незадовільно)	Не зараховано	FX
1-34			F

Рис. 4.1 – Таблиця переведення оцінки з дисциплін

Відповідно до цієї таблиці сформуємо відповідні класи еквівалентностей для системи формування оцінки. Відповідно до даної методики необхідно розбити

множину значень вхідних даних на кінцеве число підмножин (які будуть називатися класами еквівалентності), щоб кожний тест, що є представником певного класу, був еквівалентним будь-якому іншому тесту цього класу. Два тести є еквівалентними, якщо вони виявляють ті самі помилки.

Проектування тестів за методом класів еквівалентності проводиться у два етапи:

- виділення за специфікацією класів еквівалентності;

- побудова множини тестів.

На першому етапі відбувається вибір зі специфікації кожної вхідної умови та розбиття її на дві або більше групи, що відповідають так званим «правильним» класам еквівалентності (ПКЕ) та «неправильним» класам еквівалентності (НКЕ), тбто множинам допустимих для програми й недопустимих значень вхідних даних. Цей процес залежить від вигляду вхідної умови. Наприклад: якщо вхідна умова описує множину ($|x| \leq 0.5$), то визначається один ПКЕ ($-0.5 \leq x \leq 0.5$) і два НКЕ ($x < -0.5$; $x > 0.5$).

Позначимо оцінку з дисципліни $О_ц$, значення якої може бути тільки цілим позитивним числом. Визначимо КЕ відповідно до рис.4.1:

ПКЕ (Національна шкала, залік):

($60 \leq О_ц \leq 100$);

НКЕ (Національна шкала, залік):

($О_ц < 60$; $О_ц > 100$);

ПКЕ (Національна шкала, екзамен):

($0 < О_ц \leq 59$; $60 \leq О_ц \leq 74$; $75 \leq О_ц \leq 89$; $90 \leq О_ц \leq 100$);

НКЕ (Національна шкала, екзамен):

($О_ц < 0$; $О_ц > 100$);

ПКЕ (Шкала ЄКТС):

($0 < О_ц \leq 59$; $60 \leq О_ц \leq 74$; $75 \leq О_ц \leq 89$; $90 \leq О_ц \leq 95$; $96 \leq О_ц \leq 100$;);

НКЕ (Шкала ЄКТС):

($О_ц < 0$; $О_ц > 100$).

На другому етапі методу класів еквівалентності виділені класи використовуються для побудови тестів. Для НКЕ тести проектуються таким чином, що кожен тест покриває один і тільки один НКЕ, доки всі НКЕ не будуть покриті.

Проаналізувавши всі НКЕ прийшли висновку, що НКЕ для усіх ПКЕ співпадають. Разом з тим слід зазначити, що у відповідності до умов переведення оцінки між різними шкалами можуть бути визначені додаткові ПКЕ та НКЕ:

Національна шкала, екзамен:

$ПКЕ_5$: $(90 \le О_ц \le 100)$;

$НКЕ_5$: $(О_ц \le 89; О_ц > 100)$;

$ПКЕ_4$: $(75 \le О_ц \le 89)$;

$НКЕ_4$: $(О_ц \le 74; О_ц > 89)$;

$ПКЕ_3$: $(60 \le О_ц \le 74)$;

$НКЕ_3$: $(О_ц \le 59; О_ц > 74)$;

$ПКЕ_2$: $(1 \le О_ц \le 59)$;

$НКЕ_2$: $(О_ц \le 0; О_ц > 59)$;

Шкала ЄКТС:

$ПКЕ_A$: $(96 \le О_ц \le 100)$;

$НКЕ_A$: $(О_ц \le 95; О_ц > 100)$;

$ПКЕ_B$: $(90 \le О_ц \le 95)$;

$НКЕ_B$: $(О_ц \le 89; О_ц > 95)$;

$ПКЕ_C$: $(75 \le О_ц \le 89)$;

$НКЕ_C$: $(О_ц \le 74; О_ц > 89)$;

$ПКЕ_D$: $(66 \le О_ц \le 74)$;

$НКЕ_D$: $(О_ц \le 65; О_ц > 74)$;

$ПКЕ_E$: $(60 \le О_ц \le 65)$;

$НКЕ_E$: $(О_ц \le 59; О_ц > 65)$;

$ПКЕ_F$: $(35 \le О_ц \le 59)$;

$НКЕ_F$: $(О_ц \le 34; О_ц > 59)$;

$ПКЕ_{FX}$: $(1 \le О_ц \le 34)$;

$НКЕ_{FX}$: $(О_ц \le 0; О_ц > 34)$.

На наступному етапі відповідно до методики тестування чорного ящика всі неправильні дії ПЗ буде поділено на 5 моделей помилок: функціональні помилки, помилки в структурі даних, помилки інтерфейсу, помилки ініціалізації та помилки продуктивності.

Нехай в системі відсутні помилки, пов’язані з іншими видами крім продуктивності системи. Відповідно до виконаних припущень узагальнена форма проведення тестування буде мати наступний вигляд (табл.4.1):

Таблиця 4.1 – Форма тестування $ПКЕ_с$

Помилка	Оцінка помилки
Введення оцінки	0
Помилка продуктивності	10
Діапазон значень екзаменаційної оцінки	$75 \leq О_ц \leq 89$
Очікуваний результат	С

Зведена таблиця результатів тестування наведена нижче (табл.4.2):

Таблиця 4.2 – Зведена форма результатів тестування $ПКЕ_с$

Помилка	Оцінка помилки
Функціональна	0
Структури даних	0
Інтерфейсу	0
Ініціалізації	0
Продуктивності	10
Сумарна помилка	10
Діапазон значень екзаменаційної оцінки	$75 \leq О_ц \leq 89$
Очікуваний результат	С

Зведемо всі тестові випадки в єдину форму (табл. 4.3). Таблиця 4.3 показує, що набір даних, на підставі яких було проведено тестування ПЗ методом чорного ящика з урахуванням поділу на класи еквівалентностей, дозволяє визначити результат

тестування ПЗ у вигляді Correct, якщо результат тестування очікуваний, та InCorrect, якщо результат тестування непередбачений.

Таблиця 4.3 – Результати тестування визначеного набору даних

№	*Форма тесту*	*Номер тесту*	*Тестувальник*	*Категорія помилки*	*Сумарна помилка*	*Дефект*	*Результат*
1	1	1	1	1	10	1	Correct
2	1	2	1	2	20	1	Correct
3	1	3	1	3	30	1	Correct
4	1	4	1	4	40	1	Correct
5	1						
…	…	…	…	…	…	…	…
196	40	1	1	0	0	0	InCorrect
197	40	2	1	0	0	0	InCorrect
198	40	3	1	0	0	0	InCorrect
199	40	4	1	0	0	0	InCorrect
200	40	5	1	0	0	0	InCorrect

Проаналізувавши результати, наведені в таблиці 4.3 можна дійти висновків, що набір даних, отриманий в результаті тестування програмного забезпечення методом чорного ящика на підставі поділу вхідних даних на класи еквівалентностей означає, що результати тесту мають ті самі результати, що і результати тесту на дефекти, тоді як неправильне значення, отримані в результаті тестування тестером –не те саме, що результати визначення дефектів.

Виконаємо нормалізацію значень набору даних, який будемо використовувати для навчання нейронної мережі. При нормалізації значень на наборі даних проводиться навчання для процесу перетворення даних тестування в дані діапазону від 0,1 до 0,9, оскільки використовується функція активації, де значення функції ніколи не досягає 0 або 1, а саме [11]:

$$X' = \frac{0{,}8*(X-a)}{b-a} + 0{,}1\ , \qquad (5)$$

де a – номер спроби, b – номер тесту. Результат нормалізації наведено в табл. 4.4.

Таблиця 4.4 – Набір даних результатів нормалізації

Форма тесту	*Номер тесту*	*Тестер*	*Категорія помилки*	*Сумарна помилка*	*Дефект*	*Результат*
1	1	0,12	0,12	0,3	0,12	Correct
1	2	0,12	0,14	0,5	0,12	Correct
1	3	0,12	0,16	0,7	0,12	Correct
1	4	0,12	0,18	0,9	0,12	Correct
1	5	0,12	0,1	0,1	0,1	InCorrect
…	…	…	…	…	…	…
40	5	0,1	0,1	0,1	0,1	Correct
40	2	0,1	0,1	0,1	0,1	Correct
40	3	0,1	0,1	0,1	0,1	Correct
40	4	0,1	0,1	0,1	0,1	Correct
40	5	0,1	0,1	0,1	0,1	Correct

Набір даних для прогнозування точності результатів тестування ПЗ методом чорного ящика та технікою поділу на класи еквівалентності з використанням алгоритму ШНМ наведено у таблиці 4.5, значення средньоквадратичної помилки складає 0,00142438.

Таблиця 4.5 – Таблиця значень одиниці виходу Y_k з ШНМ

Y_k							
$Z_{in\ 1\text{-}1}$	0,3394	$Z_{in\ 2\text{-}1}$	0,5384	$Z_{in\ 3\text{-}1}$	0,2104	$Z_{in\ 4\text{-}1}$	0,6388
$Z_{in\ 1\text{-}2}$	0,442	$Z_{in\ 2\text{-}2}$	0,4574	$Z_{in\ 3\text{-}2}$	0,0498	$Z_{in\ 1\text{-}2}$	0,7774
$Z_{in\ 1\text{-}3}$	0,5446	$Z_{in\ 2\text{-}3}$	0,3764	$Z_{in\ 3\text{-}3}$	-0,1108	$Z_{in\ 4\text{-}3}$	0,9608
…	…	…	…	…	…	…	…
$Z_{in\ 1\text{-}104}$	0,5446	$Z_{in\ 2\text{-}104}$	0,3764	$Z_{in\ 3\text{-}104}$	-0,1108	$Z_{in\ 4\text{-}104}$	1,0616
$Z_{in\ 1\text{-}105}$	0,442	$Z_{in\ 2\text{-}105}$	0,4574	$Z_{in\ 3\text{-}105}$	0,0498	$Z_{in\ 4\text{-}105}$	0,8782

Зміни у ваговому коефіцієнті значення ваги прихованого шару ШНМ обчислюється у відповідності до формули 4.2 та представлено в таблиці 4.6.

$$V'_{ij} = V_{ij} + \Delta_{ij} \qquad (6)$$

Таблиця 4.6 – Таблиця зміни значень вагових коефіцієнтів

$V_{1\text{-}1}$	0,0000276	$V_{1\text{-}2}$	0,0000144	$V_{1\text{-}3}$	0,000096	$V_{1\text{-}4}$	0,000012
$V_{2\text{-}1}$	0,0000276	$V_{2\text{-}2}$	0,0000168	$V_{2\text{-}3}$	0,00016	$V_{2\text{-}4}$	0,000012
$V_{3\text{-}1}$	0,0000276	$V_{3\text{-}2}$	0,0000192	$V_{3\text{-}3}$	0,000224	$V_{3\text{-}4}$	0,000012
$V_{4\text{-}1}$	0,0000276	$V_{4\text{-}2}$	0,0000216	$V_{4\text{-}3}$	0,000288	$V_{4\text{-}4}$	0,000012
$V_{5\text{-}1}$	0,0000276	$V_{5\text{-}2}$	0,000012	$V_{5\text{-}3}$	0,000032	$V_{5\text{-}4}$	0,00001
$V_{6\text{-}1}$	0,0000276	$V_{6\text{-}2}$	0,000012	$V_{6\text{-}3}$	0,000032	$V_{6\text{-}4}$	0,00001
$V_{7\text{-}1}$	0,0000276	$V_{7\text{-}2}$	0,000012	$V_{7\text{-}3}$	0,000032	$V_{7\text{-}4}$	0,00001

4.2 Проведення експериментів

Після завершення навчання ШНМ на визначеному наборі даних наступним етапом є тестування набору даних. Після навчання алгоритм нейронної мережі зможе передбачити рівень точності між вихідним значенням та дефектом набору даних прогнозування та вихідним значенням дефекту набору даних реальності. При цьому використовується поняття епохи – відбулася одна епоха (epoch) – увесь датасет пройшов через нейронну мережу тільки один раз.

Нижче наведено коефіцієнти помилок результату та визначення дефекту прогнозування у порівнянні з реальними даним, за умови, що існує безліч прихованих 4 та 5 рівнів тестів, а коефіцієнт навчання ШНМ складає 0,1. В таблиці 4.7 зосереджено результати дослідження набору даних за результатами прогнозування ШНМ з 4 прихованими шарами, кількість епох – 900, швидкість навчання – 0,1 та кількість даних для тестування – 20 (20% від обсягу навчальної виборки). Графічне подання результатів дослідження за вказаними вище умовами наведено на рис.4.2.

В таблиці 4.8 наведено результати дослідження ще одного набору даних для ШНМ з 4 прихованими шарами, кількість епох збільшено до 1000, швидкість навчання – 0,1 та кількість даних для тестування – 20 (20% від обсягу навчальної виборки). Графічна інтерпретація результатів дослідження наведено на рис.4.3.

Таблиця 4.7 – Зведена таблиця порівняння результатів для 900 епох

Рядок	*Форма*	*Реальні дані*		*Прогноз*		*Співпадіння*	*Імовірність*
		Дефект	*Ціль*	*Дефект*	*Вихід*		
0	1	0	InCorrect	0,000812	InCorrect	Так	0,999188
8	2	0	InCorrect	0,000812	InCorrect	Так	0,999188
13	3	1	Correct	0,999903	Correct	ОК	0,999903
18	4	0	InCorrect	0,000813	InCorrect	ОК	0,999187
22	5	0	InCorrect	0,000813	InCorrect	ОК	0,999187
31	7	0	InCorrect	0,000813	InCorrect	ОК	0,999187
32	7	1	Correct	0,999908	Correct	ОК	0,999908
37	8	0	InCorrect	0,000813	InCorrect	ОК	0,999187
48	10	0	InCorrect	0,000812	InCorrect	ОК	0,999188
51	11	1	Correct	0,999908	Correct	ОК	0,999908
56	12	1	Correct	0,999908	Correct	ОК	0,999908
62	13	1	Correct	0,999908	Correct	ОК	0,999908
66	14	0	InCorrect	0,000813	InCorrect	ОК	0,999187
72	15	1	Correct	0,999908	Correct	ОК	0,999908
75	16	0	InCorrect	0,000855	InCorrect	ОК	0,999145
80	17	1	Correct	0,999901	Correct	ОК	0,999901
85	18	1	Correct	0,999904	Correct	ОК	0,999904
89	18	1	Correct	0,999908	Correct	ОК	0,999908
96	20	1	Correct	0,999908	Correct	ОК	0,999908
100	21	1	Correct	0,999884	Correct	ОК	0,999884

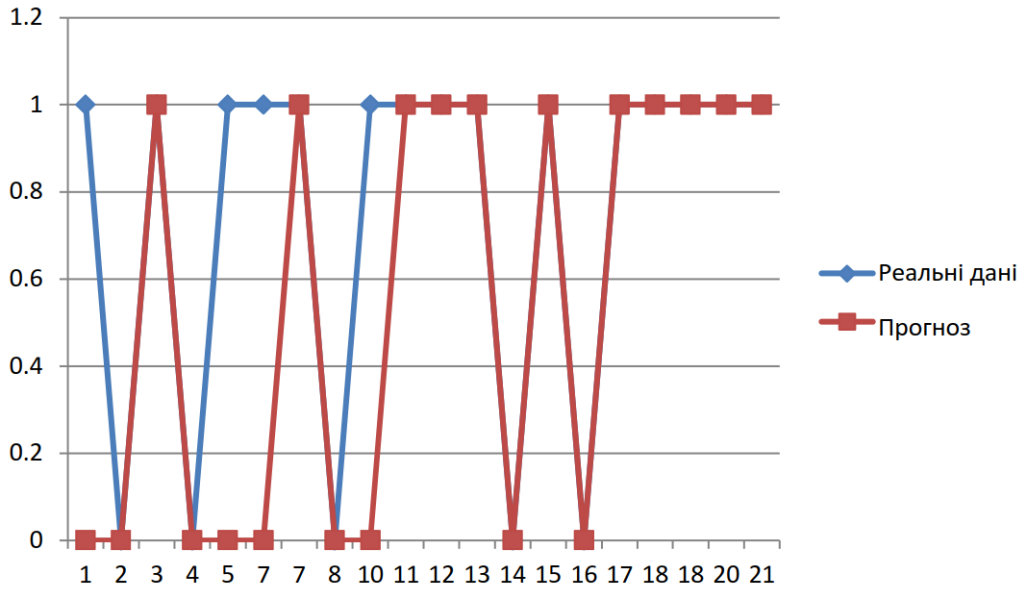

Рис. 4.2 – Графік порівняння результатів

Таблиця 4.8 – Зведена таблиця порівняння результатів для 1000 епох

Рядок	*Форма*	*Реальні дані*		*Прогноз*		*Співпадіння*	*Імовірність*
		Дефект	*Ціль*	*Дефект*	*Вихід*		
0	1	1	Correct	0,999891	Correct	OK	0,999891
8	2		InCorrect	0,000856	InCorrect	OK	0,999144
13	3	1	Correct	0,999885	Correct	OK	0,999885
18	4		InCorrect	0,000855	InCorrect	OK	0,999145
22	5	1	Correct	0,999885	Correct	OK	0,999885
31	7	1	Correct	0,999890	Correct	OK	0,99989
32	7	1	Correct	0,999891	Correct	OK	0,999891
37	8		InCorrect	0,000855	InCorrect	OK	0,999145
48	10	1	Correct	0,999891	Correct	OK	0,999891
51	11	1	Correct	0,999882	Correct	OK	0,999882
56	12	1	Correct	0,999890	Correct	OK	0,99989
62	13	1	Correct	0,999891	Correct	OK	0,999891
66	14		InCorrect	0,000853	InCorrect	OK	0,999147
72	15	1	Correct	0,999891	Correct	OK	0,999891
75	16		InCorrect	0,000855	InCorrect	OK	0,999145
80	17	1	Correct	0,999884	Correct	OK	0,999884
85	18	1	Correct	0,999890	Correct	OK	0,99989
89	18	1	Correct	0,999881	Correct	OK	0,999881
96	20	1	Correct	0,999889	Correct	OK	0,999889
100	21	1	Correct	0,999890	Correct	OK	0,99989

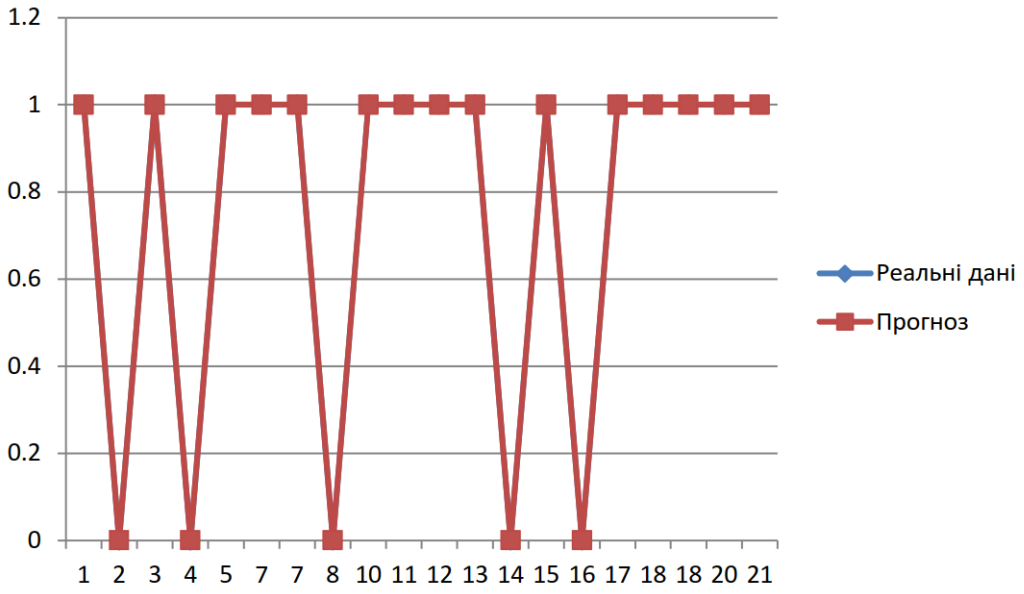

Рис. 4.3 – Графік порівняння результатів

За отриманими результатами можна зробити висновки, що рівень точності результатів, отриманих з використанням ШНМ при проведенні тестування програмного забезпечення методом чорного ящика має точність близько 85%. Для оцінювання використовувалась характеристика AUC (Area Under Curve), яка є агрегованою характеристикою якості класифікації, що не залежить від співвідношення цін помилок. Чим більше значення AUC, тим «краще» модель класифікації. Даний показник часто використовується для порівняльного аналізу декількох моделей класифікації.

Разом з тим слід зазначити, що збільшення кількості прихованих шарів ШНМ під час проведення експериментів негативно впливав на результати, про що свідчить вміст таблиці 4.9.

Таблиця 4.9 – Залежність результатів від кількості епох та шарів ШНМ

Кількість прихованих шарів ШНМ	*Кількість епох*	*Швидкість навчання*	*Точність*	*AUC*
4	900	0,1	85,00%	0,850 +/- 0,071
4	1000	0,1	99,00%	0,976 +/- 0,071
5	1000	0,1	80,00%	0,800 +/- 0,071

На рис.4.4 зображено архітектуру дизайну ШНМ з 4 прихованими шарами, 5 вхідними шарами та 1 вихідним шаром. Побудову архітектури мережі було виконано з використанням розробки, опис якої наведено в [35].

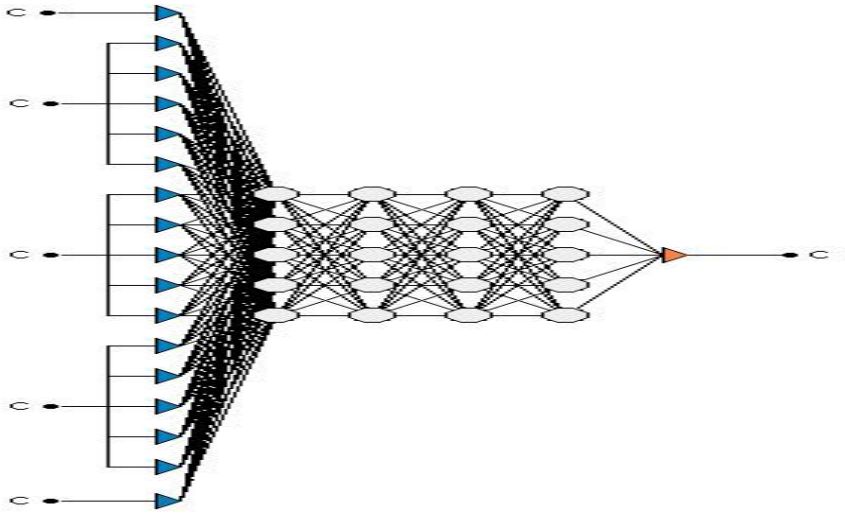

Рис. 4.4 – Архітектура ШНМ

Результати дослідження були неодноразово представлені та обговорені на міжнародних науково-практичних конференціях [36–39]. За результатами дослідження можна зробити наступні висновки: запропонований метод проведення тестування методом чорного ящика з використанням алгоритмів ШНМ є перспективним та потребує подальшого вдосконалення та розвитку. Використання моделей прогнозування наявності помилок в ПЗ для методу тестування чорного ящика, заснований на точності алгоритму нейронної мережі, може знайти своє використання для прискорення процесу тестування та підвищення якості тестування загалом.

ВИСНОВКИ

Під час підготовки атестаційної роботи були розглянуті актуальні питання вдосконалення методів тестування програмного забезпечення за рахунок використання алгоритмів штучного інтелекту. В роботі були детально досліджені різні підходи щодо використання нейромережевих технологій для тестування програмного забезпечення, проаналізовано підходи науковців та розробників програмного забезпечення до оцінювання ролі, яку відіграють алгоритми ШІ в тестуванні ПЗ.

В результаті написання атестаційної роботи за темою дослідження було проведено: аналіз першоджерел; виконано дослідження методів та алгоритмів тестування програмного забезпечення; визначено можливі шляхи покращення якості тестування за рахунок використання нейромережевих підходів як для генерації тестів, так і для формування тестових наборів даних; виконано постановку задачі дослідження.

В роботі проведено дослідження та аналіз методів, засобів і технологій, що застосовуються під час роботи з даними для їх кластеризації та класифікації. Особливу увагу в роботі приділено розробкам, що були представлені протягом останнього року. Так, наприклад, фахівці з автоматизації фірми Eggplant в жовтні 2020 року запустили нову платформу для тестування програмного забезпечення на базі штучного інтелекту. Особливості представленої платформи Digital Automation Intelligence (DAI) Eggplant [36]:

- наскрізна автоматизація на основі хмари;

- моніторинг: додавання точок даних і показників розширеного взаємодії з користувачем (UX), що дозволяє клієнтам порівнювати продуктивність UX своїх додатків.

На підставі проведеного аналізу предметної області в атестаційній роботі запропоновано методику проведення функціонального тестування ПЗ з використанням нейромережевих технологій – використання штучних нейронних

мереж для підтримки прийняття рішень множинних інтелектуальних агентів, що відстежують проведення тестування. З цією метою в атестаційній роботи було виконано розробку структури ШНМ, розроблено алгоритм навчання на підготовлених наборах даних, виконано експериментальні дослідження, які наочно довели ефективність використання цієї методики. Під час проведення експериментів було з’ясовано, що основний вплив на ефективність використання ШНМ при проведенні тестування ПЗ методом «чорного ящика» має кількість епох тестування, а збільшення кількості прихованих шарів навпаки, зменшує достовірність отриманих результатів, збільшуючи кількість хибних спрацювань. Запропоновану методологію було перевірено на реальному програмному додатку, в результаті чого було виявлено 2 дефекти ПЗ.

Наукові дослідження, що були покладені в основу атестаційної роботи, проводились автором самостійно та неодноразово були представлені на науково-практичних конференціях та семінарах [37-39].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Мейерс Г. Искусство тестирования. – М.: Финансы и статистика, 1982.

2. Бейзер Б. Тестирование черного ящика. Технологии функционального тестирования программного обеспечения и систем. – СПб: Питер, 2004.– 318 с.

3. Куликов С.С. Тестирование программного обеспечения. Базовый курс. Минск: Четыре четверти, 2017. – 312 с.

4. Савин, Р. Тестирование Дот Ком, или Пособие по жестокому обращению с багами в интернет-стартапах. Р. Савин. – Москва : Дело, 2007. – 312 с.

5. Макконнелл С. Совершенный код. Мастер-класс / Пер. с англ. — М.: Издательство «Русская редакция», 2010. — 896 стр.

6. Khan, M. E., & Khan, F. (2012). A comparative study of white box, black box and grey box testing techniques. Int. J. Adv. Comput. Sci. Appl, 3(6). URL: https://dx.doi.org/10.14569/IJACSA.2012.030603 (дата звернення: 19.09.2020).

7. Crispin, L., 2019. Blog - Practical use cases for machine learning. s.l.:Mabl. Diffblue, 2019. Diffblue - AI for Code. [Online] URL: https://www.diffblue.com/about-us

8. Haenlein, Michael & Kaplan, Andreas. (2019). A Brief History of Artificial Intelligence: On the Past, Present, and Future of Artificial Intelligence. California Management Review. URL: https://www.researchgate.net/publication/334539401_A_Brief_History_of_Artificial_Intelligence_On_the_Past_Present_and_Future_of_Artificial_Intelligence (дата звернення: 06.09.2020).

9. Groz, R., Simao, A., Bremond, N., & Oriat, C. (2018). Revisiting AI and testing methods to infer FSM models of black-box systems. In IEEE/ACM 13th Int. Work. on Automation of Software Test, (pp.16-19). URL: https://repositorio.usp.br/item/002900234 (DOI: 10.1145/3194733.3194736) (дата звернення: 08.09.2020).

10. Barr, E. T., Harman, M., McMinn, P., Shahbaz, M., & Yoo, S. (2014). The oracle problem in software testing: A survey. IEEE transactions on software engineering, 41(5), (pp:507-525).URL: http://www0.cs.ucl.ac.uk/staff/M.Harman/tse-oracle.pdf (дата звернення: 18.09.2020).

11. Neural Networks: A Comprehensive Foundation, by Simon HAYKIN; Macmillan College Publishing~ New York, USA; IEEE Press, New York, USA; IEEE Computer Society Press, Los Alamitos, CA, USA; 1994; 696 pp. – URL: http://amutiara.staff.gunadarma.ac.id/Downloads/files/14638/Neural_Networks._A_Compre_Found__2nd_.pdf (дата звернення: 29.10.2020).

12. Splunk, 2019. Artificial Intelligence and Machine Learning | Splunk. [Online] URL: https://www.splunk.com/en_us/explore/machine-learning-artificial-intelligence.html

13. Николенко С., Кадурин А., Архангельская Е. Глубокое обучение. – СПб: Питер, 2018. – 480 с.

14. A. Yerokhin, O. Zolotukhin. Fuzzy Probabilistic Neural Network in Document Classification Tasks. Interbranch collection of scientific papers «Information Extraction and Processing». National Academy of Sciences of Ukraine. -2019. http://vidbir.ipm.lviv.ua/

15. Yasnitsky L.N. (2020). Whether Be New “Winter” of Artificial Intelligence?. In: Antipova T. (eds) Integrated Science in Digital Age. ICIS 2019. Lecture Notes in Networks and Systems, vol 78. Springer, Cham. URL: https://link.springer.com/chapter/10.1007%2F978-3-030-22493-6_2 (дата звернення: 22.09.2020).

16. Meinke, K. & Bennaceur, A., 2018. Machine Learning for Software Engineering: Models, Methods, and Applications. Gothenburg, Sweden, IEEE. URL: https://www.researchgate.net/publication/321807797_Machine_Learning_for_Software_Engineering_Models_Methods_and_Applications (дата звернення: 20.10.2020).

17. Godefroid, P., Klarlund, N., & Sen, K. (2005). DART: directed automated random testing. In Proc. of 2005 ACM SIGPLAN conference on Programming language design and implementation (pp. 213-223). URL: https://dl.acm.org/doi/10.1145/1065010.1065036 (дата звернення: 19.09.2020).

18. Chi, Z., Xuan, J., et al. (2017). Multi-level random walk for software test suite reduction. IEEE Comp. Intelligence Magazine, 12(2), (pp.24-33). URL: https://ieeexplore.ieee.org/document/7895279 (дата звернення: 22.09.2020).

19. Sahin, O., & Akay, B. (2016). Comparisons of metaheuristic algorithms and fitness functions on software test data generation. Applied Soft Computing, 49, 1202-1214.

URL: https://dl.acm.org/doi/abs/10.1016/j.asoc.2016.09.045 (дата звернення: 22.09.2020).

20. Smilgyte K. Artificial Neural Networks Application in Software Testing Selection Method / K.Smilgyte, J. Nenortaite // Hybrid Artificial Intelligent Systems. - 2011. - Vol. 6678. - P.247-254. URL: https://link.springer.com/chapter/10.1007/978-3-642-21219-2_32 (дата звернення: 24.09.2020).

21. King, T. M. et al., 2019. AI for Testing Today and Tomorrow: Industry Perspectives. Newark, CA, USA, IEEE. URL: https://www.researchgate.net/publication/333229220_AI_for_Testing_Today_and_Tomorrow_Industry_Perspectives (дата звернення: 7.10.2020).

22. King, T., 2019. Masterclass: AI and Machine Learning Skills for the Testing World. [Online] URL: https://youtu.be/gvHnbexnn-4 (дата звернення: 7.10.2020).

23. AISTA. Artificial Intelligence for Software Testing Association. – Назва з екрану. [Online] URL: https://www.aitesting.org/ (дата звернення: 7.10.2020).

24. Arbon, J., 2018. AI for Software Testing - AICamp. [Online] URL: https://youtu.be/eit9DYa_DMg (дата звернення: 13.10.2020).

25. Kraus, D. (2018). Machine Learning and Evolutionary Computing for GUI-based Regression Testing. arXiv preprint arXiv:1802.03768. (дата звернення: 18.09.2020).

26. Marijan, D., Gotlieb, A. & Ahuja, M. K., 2019. Challenges of Testing Machine Learning Based Systems. Newark, CA, USA, USA, IEEE. URL: https://pdfs.semanticscholar.org/fe01/04b0809e720b9f423d3ce278e03b9a5fea4d.pdf

27. Marijan, D., Gotlieb, A. & Ahuja, M. K., 2019. Challenges of Testing Machine Learning Based Systems. Newark, CA, USA, USA, IEEE. https://www.pre-crime.eu/techreps/TR-Precrime-2020-03.pdf (дата звернення: 18.10.2020).

28. Johnson, R. C., 2018. Overcoming AI Bias with AI Fairness. [Online] URL: https://cacm.acm.org/news/233224-overcoming-ai-bias-with-ai-fairness/fulltext

29. Gerrard, P., 2017. The New Model for Testing. [Online] URL: https://youtu.be/1Ra1192OpqY (дата звернення: 20.10.2020).

30. Computing, A. (2006). An architectural blueprint for autonomic computing. IBM White Paper, 31, 1-6. URL: https://www-03.ibm.com/autonomic/pdfs/AC Blueprint White Paper V7.pdf (дата звернення: 20.10.2020).

31. Arbon, J., 2019. The AI Testing Singularity | STAREAST 2019. [Online] URL: https://www.youtube.com/watch?v=9xLnHcMIhhk (дата звернення: 18.10.2020).

32. Functionize, 2019. Automation Testing with Machine Learning - Functionize. [Online] URL: https://www.functionize.com/ (дата звернення: 22.10.2020).

33. Lachmann, R., 2018. Machine Learning-Driven Test Case Prioritization Approaches for Black-Box Software Testing. Nuremberg, Germany, European Test and Telemetry Conference ettc2018. URL: https://www.ama-science.org/proceedings/getFile/ZwtmZt== (дата звернення: 22.10.2020).

34. Положення про організацію освітнього процесу в Харківському національному університеті радіоелектроніки. [Електронній ресурс] URL: https://nure.ua/wp-content/uploads/Main_Docs_NURE/polozhennja-pro-organizaciju-osvitnogo-procesu-v-hnure.pdf (дата звернення: 16.10.2020).

35. Maksym Bekuzarov, Oleksandr Samantsov, Oksana Mazurova,Mariia Shirokopetleva. Neural Network Architecture Editor With Code Generation. Problem of Infocommunications. Science and Technology (PIC S&T′2020), Kharkiv, Ukraine.- 6-9 October 2020.

36. Turevska, O. , Shubin, I. Improving the automated testing of Web-based services by reflecting the social habits of target audiences. 2015 Information Technologies in Innovation Business Conference, ITIB 2015 - Proceedings, 2015, с. 93-96

37. О.Ф.Лановий, О.В.Золотухін. Застосування нейромережевого підходу для класифікації втручань в роботу комп'ютерних систем // Застосування інформаційних технологій у діяльності НПУ: матеріали наук.-практ. семінару (м.Харків, 21 грудня 2018 р.) / МВС України, Харк.нац.ун-т внутр.справ. Харків. ХНУВС, 2018.– С.78-79.

38. Лановий О.Ф. Про один підхід до функціонального тестування web-додатків // Полиграфические, мультимедийные и web-технологии. Т1. Тез. докл. 2-й Международ. науч.-техн. конф.(16-22 мая 2017) / редкол.: ВФ Ткаченко, ИБ Чеботарева и др.–Харьков: ХНУРЭ, 2017.–246 с.

39. Лановий О.Ф. Візуалізація в методах тестування програмного забезпечення // Харків, 6-а Міжнародна науково-технічна конференція «ІСТ-2017», ХНУРЕ, 11-16 вересня 2017 р., С.110-111.

40. TestCraft, 2020. Codeless Selenium Test Automation with AI-Based Maintenance | TestCraft. [Online] URL: https://www.testcraft.io/#about (дата звернення: 2.12.2020).