Вінницький національний технічний університет
(повне найменування вищого навчального закладу)
Факультет інформаційних технологій та комп'ютерної інженерії
(назва факультету (відділення))
Кафедра програмного забезпечення
(повна назва кафедри (предметної, циклової комісії))

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка методів і програмного засобу для системи адаптивного тестування знань»

Виконав: студент 2-го курсу, групи 2ПІ-22м спеціальності 121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Ситніков Є. О.
(прізвище та ініціали)
Керівник: д.т.н., проф. каф. ПЗ
Ліщинська Л. Б.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
«11» грудня 2023 р.
Опонент: к.т.н., доц. каф, ЗІ
Дудатьєв А. В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
«11» грудня 2023 р.

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н., проф., Романюк О. Н.
«11» грудня 2023 р.

Вінниця ВНТУ - 2023 рік

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти ІІ-й (магістерський)
Галузь знань 12 Інформаційні системи
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
« 19 » вересня 2023 р.

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Ситнікову Євгенію Олександровичу

1. Тема роботи: Розробка методів і програмного засобу для системи адаптивного тестування знань
керівник роботи: д.т.н., проф. кафедри ПЗ Ліщинська Л. Б., затверджені наказом вищого навчального закладу від « 18 » вересня 2023 р. № 247.

2. Строк подання студентом роботи: « 5 » грудня 2023 року.

3. Вихідні дані до роботи: середовища розробки – Microsoft Visual Studio 2022, Weka, SQL Server Management Studio, мова розробки – C#, операційна система – Windows; методи навчання моделі штучного інтелекту – класифікаційні; алгоритми навчання моделі – дерево рішень, найближчих сусідів, класифікатор Байєса.

4. Зміст текстової частини: вступ; аналіз особливостей адаптивного тестування, розробка алгоритмів та методів для покращення систем адаптивного тестування; підготовка та навчання моделі штучного інтелекту; розробка серверного та клієнтського додатків; тестування системи; економічна частина; висновки; список використаних джерел; додатки.

5. Перелік графічного матеріалу: графіки алгоритмів адаптивності;

результати підготовки, навчання, тестування, перенавчання моделі штучного інтелекту; UML- діаграми проектування серверної частини додатку; графічний інтерфейс клієнтського додатку; тестування додатку.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	виконання прийняв
1–5	д.т.н., проф. каф. ПЗ Ліщинська Л.Б.	19.09.2023	05.12.2023
6	к.е.н., доц. каф. ЕПВМ Кавецький В. В.	20.11.2023	24.11.2023

7. Дата видачі завдання « 19 » вересня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної галузі та вимог до створення системи	20.09 – 11.10	Вик
2	Проектування програмного додатку	11.10 – 18.10	Вик
3	Підготовка та навчання моделі ШІ	18.10 – 01.11	Вик
4	Обгрунтування засобів для розробки додатку	01.11 – 03.11	Вик
5	Розробка програмного засобу системи	03.11 – 20.11	Вик
6	Економічна частина	20.11 – 24.11	Вик
7	Оформлення матеріалів до захисту МКР	24.11 – 01.12	Вик

Студент (підпис) **Ситніков Є. О.** (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи (підпис) **Ліщинська Л. Б.** (прізвище та ініціали)

АНОТАЦІЯ

УДК 681.03

Ситніков Є. О. Розробка методів і програмного засобу для системи адаптивного тестування знань. Магістерська кваліфікаційна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2023. 102 с.
На укр. мові. бібліогр.: 44 назв; рис.: 50; табл.: 8.

У магістерській кваліфікаційній роботі було проведено аналіз стану галузі та літературні джерела в області адаптивного тестування знань.

Покращено метод підбору питань та калібрування їх складності у реальному часі, який якісно відрізняється від аналогічних підходів у існуючих розробках, що дозволяє точніше формувати завдання за складністю до рівня підготовки користувача.

Розроблено метод оцінки знань користувачів за допомогою моделі штучного інтелекту на основі класифікаційного дерева, котра приймає рішення про закінчення процесу тестування, що забезпечує зменшення часу процесу тестування та економію ресурсів для його забезпечення.

Проаналізовано та обґрунтовано технічні засоби та підходи для реалізації програмного засобу для системи адаптивного тестування знань на основі розроблених та вдосконалених методів. Розроблено програмний засіб для перевірки результатів дослідження.

У економічній частині роботи було розраховано комерційний потенціал розробки, економічну ефективність науково-технічної розробки та її економічну привабливість для потенційних інвесторів.

Ключові слова: адаптивне тестування, машинне навчання, класифікація.

ABSTRACT

UDC 681.03

Sytnikov Y. O. Development of methods and software for the adaptive knowledge testing system. Master's degree work on specialty 121 - software engineering, educational program - software engineering. Vinnytsia: VNTU, 2023. 102 p.

In Ukrainian language bibliography: 44 назв; рис.: 50; табл.: 8.

An analysis of the state of the industry and literature sources in the field of adaptive knowledge testing was carried out in the master's qualification work.

The method of selecting questions and calibrating their complexity in real time has been improved, which is qualitatively different from similar approaches in existing products, which allows to form tasks more accurately according to the complexity of the user's knowledge level.

A method of assessing user knowledge using an artificial intelligence model based on a classification tree has been developed, which makes a decision on the end of the testing process, which ensures a reduction in the time of the testing process and saves resources for its provision.

The technical tools and approaches for the implementation of the software for the adaptive knowledge testing system based on the developed and improved methods were analyzed and substantiated. A software application has been developed for checking the results of the investigation.

In the economic part of the work, the commercial potential of the development, the economic efficiency of the scientific and technical development and its economic attractiveness for potential investors were calculated.

Keywords: adaptive testing, machine learning, classification.

ЗМІСТ

ВСТУП

У сучасному світі, в освіті складається ситуація, під впливом якої традиційне тестування у вигляді стандартизованих тестів фіксованою довжини, переростає в нові ефективні форми адаптивного тестування, що базується на відмінних від традиційних теоретико-методологічних підходів [1]. Основна ідея адаптивного тестування знань, що викликає увагу дослідників у сфері тестування, полягає в тому, що тестові завдання необхідно адаптувати (підігнати) по складності до рівня знань користувачів. Дослідники виходять з тих міркувань, що користувачам з низьким рівнем підготовки марно давати важкі завдання, так як з великою ймовірністю вони не зуміють їх виконати вірно. Окремі винятки із описаного вище твердження бувають, але вони вкрай небажані, оскільки призводять до зниження ефективності тестування даної категорії користувачів, сприяючи тим самим зростанню помилкового компонента в їх оцінках. Також марно пропонувати легкі завдання при тестуванні користувачів з високим рівнем знань [2]. Очевидно, що використання занадто простих завдань може призвести до того, що більшість користувачів отримають високі бали, тому оцінка буде хибна через невідповідність складності завдань до рівня користувачів. Описані проблеми ефективно вирішує адаптивне тестування знань – кожне наступне завдання вибирається із банку завдань відповідно до «рейтингу» користувача, який в процесі тестування постійно змінюється [3]. Тому розробка методів і програмних засобів для систем адаптивного тестування знань є актуальними задачами різноманітних сфер діяльності де, потрібна об’єктивна та ефективна оцінка знань.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета і завдання дослідження. Метою дослідження у роботі є підвищення ефективності тестування знань, зокрема в освітній системі за рахунок використання технологій машинного навчання.

Для досягнення поставленої мети необхідно виконати наступні завдання:

- проаналізувати предметну галузь та існуючі рішення в галузі адаптивного тестування знань;

- розробити метод для визначення моменту закінчення процесу тестування та оцінки знань користувача;

- покращити метод адаптивності системи тестування для підлаштування завдань до рівня знань користувача;

- підготувати та провести навчання моделі штучного інтелекту для застосування нового методу;

- спроектувати та розробити програмне забезпечення для адаптивного тестування знань з використанням моделі штучного інтелекту;

- Провести тестування розробленої системи.

Об’єкт дослідження – процеси адаптивного тестування знань.

Предмет дослідження – методи та засоби адаптивного тестування знань.

Методи дослідження. В процесі виконання роботи використовувались комп’ютерне моделювання для аналізу і покращення методу «адаптивності»; методи машинного навчання для розробки моделі класифікації користувачів на основі алгоритмів: дерева рішень, пошуку найближчих сусідів, класифікатора Байєса; методи архітектурного проектування програмних засобів за допомогою UML діаграм та кодогенерації інструментальним програмним забезпеченням, методи розробки сервісно-орієнтованих програмних засобів, принципи об’єктно-орієнтованого програмування для розробки програмного засобу для адаптивного тестування знань, методи тестування ПЗ для перевірки коректності отриманих теоретичних та практичних результатів.

Гіпотеза дослідження полягає в тому, що технології машинного навчання здатні підвищити якість та швидкість контрольно-оціночного процесу знань.

Наукова новизна отриманих результатів:

1. Розроблено метод оцінки знань користувача за допомогою моделі класифікаційного дерева, котра приймає рішення про закінчення тестування та визначає кінцевий результат тестування, що забезпечує зменшення часу процесу

тестування та, відповідно, економію ресурсів для його забезпечення;

2. Подальшого розвитку отримав метод «адаптивності» комп'ютерного тестування знань за рахунок динамічного підбору коефіцієнтів зміни складності завдань, що дозволило більш точно формувати завдання за складністю, відповідно до рівня підготовки користувача, та калібрувати тестові завдання.

Практична цінність отриманих результатів полягає в тому, що на основі запропонованих теоретичних положень та результатів досліджень розроблено та покращено методи та програмний засіб для адаптивного тестування знань.

Особистий внесок здобувача. Всі наукові результати, викладені у магістерській кваліфікаційній роботі, отримані автором особисто. У наукових працях, опублікованих у співавторстві, автору належать: алгоритм адаптивності системи тестування знань [4], метод для класифікації оцінки та припинення процесу тестування [5].

Апробація матеріалів магістерської кваліфікаційної роботи. Основні положення магістерської кваліфікаційної роботи доповідались та обговорювались на:

- IV Міжнародній науково-практичній конференції «Інформаційні технології та комп'ютерна інженерія» (28-30 травня 2014 р., м. Вінниця, Україна);

- міжнародній науково-практичній Інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ» (20-21 листопада 2023 р., м. Вінниця, Україна).

Публікації. Результати роботи опубліковані в двох наукових працях – збірниках тез-доповідей міжнародних конференцій:

- IV міжнародній науково-практичній конференції «Інформаційні технології та комп'ютерна інженерія» (28-30 травня 2014 р., м. Вінниця, Україна) [4];

- міжнародній науково-практичній Інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ» (20-21 листопада 2023

р., м. Вінниця, Україна) [5].

1 АНАЛІЗ МЕТОДІВ І ПРОГРАМНИХ ЗАСОБІВ ДЛЯ СИСТЕМ АДАПТИВНОГО ТЕСТУВАННЯ ЗНАНЬ

1.1 Аналіз предметної галузі та вимог до створення систем адаптивного тестування знань

В будь-якій освітній системі необхідний простий і ефективний спосіб вимірювання знань учасників навчального процесу. Тільки так можна оцінити якість роботи системи освіти. Тести вважаються одним з найпростіших, надійних і об'єктивних способів оцінки знань. Останнім часом на Заході все більшою популярністю починають користуватися адаптивні тести.

Основний атрибут, який властивий тестовим завданням адаптивного тесту – це рівень складності, отриманий емпіричним шляхом. Тобто перш ніж потрапити в тестовий набір, кожне завдання проходить певну апробацію на досить великій цільової аудиторії користувачів. В загальному алгоритм адаптивного тестування знань складається з наступних кроків:

1. З набору завдань вибирається найбільш підходяще (за певними параметрами) для користувача завдання.

2. Користувач вирішує завдання правильно чи неправильно.

3. Оцінка користувача оновлюється на підставі цієї відповіді.

4. Дані кроки повторюються до тих пір, поки згідно певна умова виконується. Така умова називається критерієм зупинки тестування. Як тільки вона задовольняється тестування вважається завершеним.

Оскільки на початку процесу тестування системі невідомо про рівень знань користувача поки він не відповість щонайменше на перше запитання, тестування починається з середнього рівня складності, вважаючи рівень підготовки користувача як «середній». Наступне завдання системи – якомога швидше пристосуватись (адаптуватись) до рівня користувача, для найефективнішої оцінки його знань.

Блок-схему такого алгоритму адаптивності системи тестування наведено на рисунку 1.1.

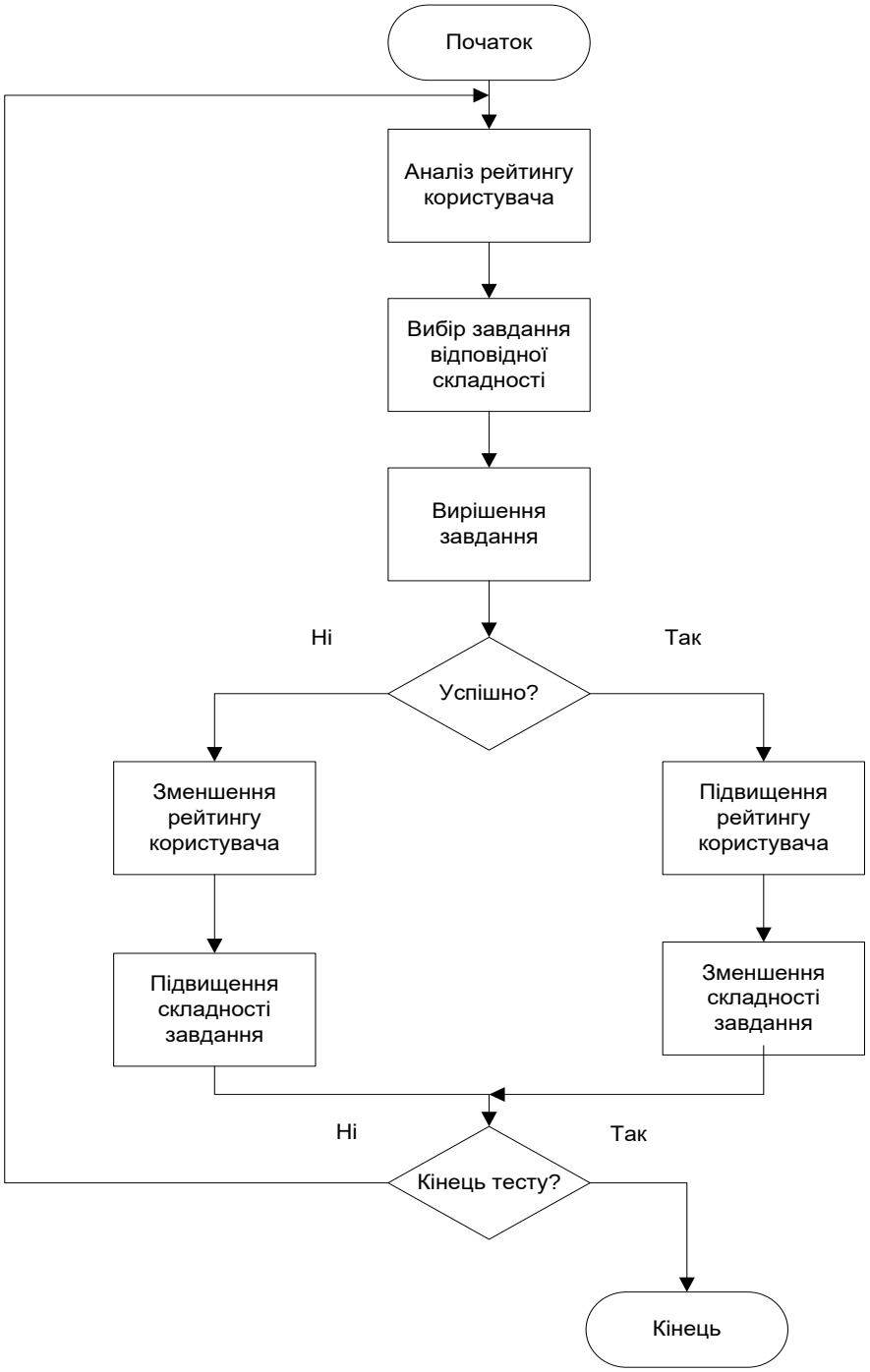


Рисунок 1.1 – Блок-схема алгоритму адаптивності системи тестування

В загальному випадку для розробки системи адаптивного тестування знань необхідні наступні компоненти:

1. Набір тестових завдань, які відкалібровані за складністю. Банк завдань повинен калібруватись відповідно до певної психометричної моделі.

2. Точка входу в тест. В основному всі системи адаптивного тестування знань припускають, що кожен користувач, який починає тестування має

«середній» рівень знань, але якщо користувач використовує систему повторно, є можливість починати тестування з іншого рівня.

3. Логіка вибору завдання з тестового набору. Кожна система тестування знань імплементує власну логіку для найбільш точного підбору завдань під рівень знань користувача.

4. Алгоритм підрахунку результатів. Після кожного вирішеного завдання, не важливо чи успішно, оцінка користувача змінюється в ту, чи іншу сторону. В результаті проходження N завдань система складає результуючу оцінку, яка, за потреби, може бути приведена до будь-якої системи оцінювання [6].

5. Критерій визначення завершення тестування. Система може пропонувати користувачеві завдання до тих пір, доки вона не зможе адекватно оцінити рівень його знань. Саме момент, коли оцінка стає «адекватною» і є таким критерієм. В кожні системі тестування даний критеріїв може кардинально відрізнятись. В класичному тестуванні, зазвичай, процес тестовая зупиняється коли досягнуто ліміту дозволених помилок. В адаптивному ж тестуванні це може бути при досягненні певної межі «рівня знань» в обидва боки. В цьому полягає одна з ключових переваг адаптивного тестування, а саме – точність оцінювання знань користувача [7, 8].

1.2 Аналіз та порівняння аналогів програмних засобів для адаптивного тестування знань

Відомими аналогами систем адаптивного тестування знань є програмне забезпечення «x-TLS», «Moodle»

Система Moodle побудована наступним чином: існує базовий тест середньої важкості на «4». При виконанні тесту на 100% тестований переходить до тесту на «5», при певній кількості неправильних відповідей – переходить на тест рівня «3». Крім дана система дозволяє встановлювати залежність тестування від проходження попередніх тестів, тобто користувач має змогу стартувати тестування не з «середнього» рівня, а з того, який вже збережений в системі. Система має велику кількість додаткових плагінів, таких як QuizPor,

Activity Locking, Drag and drop question, Image target, та багато інших

QuizPor – плагін, який можна розглядати як модуль адаптивного тестування. Він працює тільки з тестами систем тестування HotPotatoes і Qedos.

Головна перевага цього модуля – можливість встановлювати залежність від проходження інших тестів. Як і в більшості систем адаптивного тестування знань, центральним елементом тестування тут є складність завдань, і не яка-небудь визначена приблизно, а математично обґрунтована. Як вказує сам розробник, Moodle може реалізувати сценарій адаптивного тестування, але довести що це саме адаптивне тестування буде не легко [9].

Іншим відомим аналогом адаптивного тестування є x-TLS. В ній для конкретного сценарію тестування створюється три набори завдань: 1, 2 і 3 рівня складності, а також виставляється 2 границі переходу – «успіх», «провал». На кожному із рівнів, тестування ведеться або до кінця набору завдань відповідного рівня, або до досягнення границь в відсотках від набору завдань відповідного рівня складності. При досягненні границі «успіх» тестований переходить на наступний рівень і отримує 1 бал, при досягненні границі «провал» тестування завершується, рівно, як і при закінченні набору завдань, якби не було досягнуто прохідного відсотку.

Як і в більшості систем адаптивного тестування знань в системі Moodle, типи завдань, їх кількість і складність – індивідуальні. Проте вибір завдань здійснюється на основі певної величини, яка характеризує складність. Ця величина є просто числовим вираженням того, наскільки завдання оцінене укладачем тесту по певній шкалі. Саме через вплив людського фактору, достовірність результатів тестування зменшується. Часто складається ситуація коли найважчі завдання, за думкою розробника тесту, насправді мають середню складність, або навпаки. Розробники Moodle подбали, щоб складність завдань була математично обгрунтованою. Проте в даному програмному забезпеченні використовується жорстка перевірка відповіді, тобто відповідь рівна правильній, або ні. Тобто все, або нічого.

Адаптивна система тестування знань x-TLS не пропонує ефективних

механізмів коригування складності завдань. Тобто, запрограмована складність укладачем тесту, не змінюється і завжди залишається сталою величиною, обраною повністю емпірично. Цей факт не дозволяє говорити про адекватність оцінки знань такою системою [10].

Пропонується вдосконалити систему адаптивного тестування таким чином, щоб вона ефективніше робила вибір тестового завдання та могла оцінити користувача та завершити тестування швидше, ніж існуючі аналоги.

Вдосконалення системи тестування буде досягнуто за рахунок проектування та навчання моделі штучного інтелекту. Вона буде виконувати роль механізму оцінки користувача. Крім того, складність самих завдань буде змінюватись в залежності від відповідей користувачів, і в перспективі всі завдання отримають максимально адекватну складність на основі статистичної інформації.

Дане вдосконалення класичної системи адаптивного тестування знань допоможе вирішити досить важливу проблему адаптивного тестування. Складність тестового завдання перетвориться на динамічну величину і постійно буде змінюватись в процесі тестування. Тому тестові завдання будуть підлаштовуватись під певний контингент тестованих.

Ці завдання спрямовані на підвищення достовірності і ефективності тестування [11].

1.3 Постановка задачі дослідження

Система освіти та оцінювання знань розвивається швидкими темпами в більшості розвинених країн світу. Це сприяє постійним вдосконаленням та інноваціям в цій області, які в свою чергу поширюють свій вплив на всі інші сфери людського життя. Важливим елементом освітнього процесу є ефективний контроль знань. Класичні системи тестування сьогодні досить поширені, але вони втрачають свою актуальність через низку недоліків.

Для контролю знань з використанням тестових завдань слід розробити математичні моделі оцінювання знань, які повинні володіти здібностями

самонавчання для тих випадків, коли викладачеві складно підібрати параметри моделі вручну. Це непросте завдання, особливо коли кількість параметрів велика або коли викладач погано уявляє собі їх призначення. В деяких випадках процес самонавчання пройде швидше, ніж ручний підбір. У той же час параметри, підібрані в результаті самонавчання можуть бути відкоректовані [12]. Для такого самонавчання необхідно розробити метод для автоматичного коригування складності завдань. Він буде нівелювати людський чинник, який виникає у аналогічних розробках, де укладач тестів визначає жорстко встановлену складність завдання, яке для майбутніх користувачів може виявитись зовсім іншої складності: занадто простим, або занадто важким. Даний метод повинен «навчати» систему на основі досвіду її користувачів, і, врешті, всі тестові завдання будуть оцінені максимально об’єктивно.

Основою математичної моделі системи тестування знань може слугувати модель машинного навчання. Така модель здатна на основі прикладів узагальнювати отримані знання і використовувати їх при прийнятті рішень над невідомими наборами даних. Коригувати вагові коефіцієнти можна і вручну. Для вирішення поставлених завдань, математичною моделлю тестування може слугувати класифікаційна модель машинного навчання.

Функціональні вимоги розроблюваної системи наступні:

- реалізація найпоширеніших типів завдань: з однією відповіддю, з декількома відповідями, з введенням вільного тексту
- можливість додавання картинки до завдання;
- різні ролі доступу до додатку: адміністратор, користувач, екзаменатор;
- кожен користувач розпочинає тестування на «середньому» рівні складності із припущенням що його рівень знань теж «середній»;
- після проходження завдання успішно «рейтинг» користувача перераховується у більшу сторону, а «складність» пройденого завдання зменшується на мінімальну величину;
- після проходження завдання невдало «рейтинг» користувача перераховується у меншу сторону, а «складність» пройденого завдання

збільшується на таку ж величину;

- коефіцієнт коригування рейтингу користувача перераховується динамічно для максимально швидкої «підгонки» складності завдань для користувача;

- максимально можлива сума балів за проходження тесту рахується як сукупна величина балів при проходження всього тестового набору учпішно, з урахуванням що кожен користувач розпочинає тестування із «середнього» рівня складності;

- користувач не повинен проходити весь тестовий набір, тестування вважається завершеним як тільки модель оцінки здатна класифікувати користувача з вірогідністю 70% або більше;

- результуюча оцінка повинна бути у відсотковому вигляді 0-100% від максимально можливої, що дозволяє перевести її у будь-яку систему оцінювання.

Перед системою поставлені такі нефункціональні вимоги:

- зменшення кількості необхідних завдань, які пропонуються користувачеві на 20-40%;

- клієнт-серверна архітектура додатку;

- зв'язок клієнтського та серверного додатків за допомогою технології WCF;

- можливість роботи через локальну мережу та через Internet;

- реляційний провайдер бази даних для безпеки та цілісності даних – MS SQL Server;

- мова програмування серверного додатку C#;

- максимальний час опрацювання запиту серверним додатком – 300 мілісекунд для запобігання втрати користувачем часу, виділеного на проходження тесту;

- логування помилок роботи;

- тип моделі машинного навчання – класифікаційне дерево;

- технологія для розробки користувацького інтерфейсу – WPF.

1.4 Висновки

Здійснивши аналіз та класифікацію програмних засобів адаптивного тестування знань визначено, що основним недоліком більшості сучасних систем адаптивного тестування є відсутність гнучкості логіки системи. Тобто користувач проходить тест, і результат цього тестування залежить, крім рівня знань користувача, від рівня знань того, хто укладав тестові завдання. Наслідком такого тестування є можливість отримати найлегші завдання з тесту і, відповідно, найвищу оцінку, або ж навпаки. Звісно таке тестування не є об’єктивним і адекватним.

Вдосконалення системи тестування буде досягнуто за рахунок проектування та навчання моделі машинного навчання. Вона буде виконувати роль механізму оцінки користувача. Крім того, складність самих завдань буде змінюватись в залежності від відповідей користувачів, і в перспективі всі завдання отримають максимально адекватну складність на основі статистичної інформації.

2 ВДОСКОНАЛЕННЯ МЕТОДІВ ТА ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАСОБУ ДЛЯ АДАПТИВНОГО ТЕСТУВАННЯ ЗНАНЬ

2.1 Розробка методу оцінки користувача та завершення процесу тестування

Класичний алгоритм адаптивного тестування працює так, що кожне наступне завдання має складність вищу, або нижчу ніж попереднє, відповідно до успішності вирішення попереднього. Даний алгоритм можна зобразити подібно графіку зображеного на рисунку 2.1.

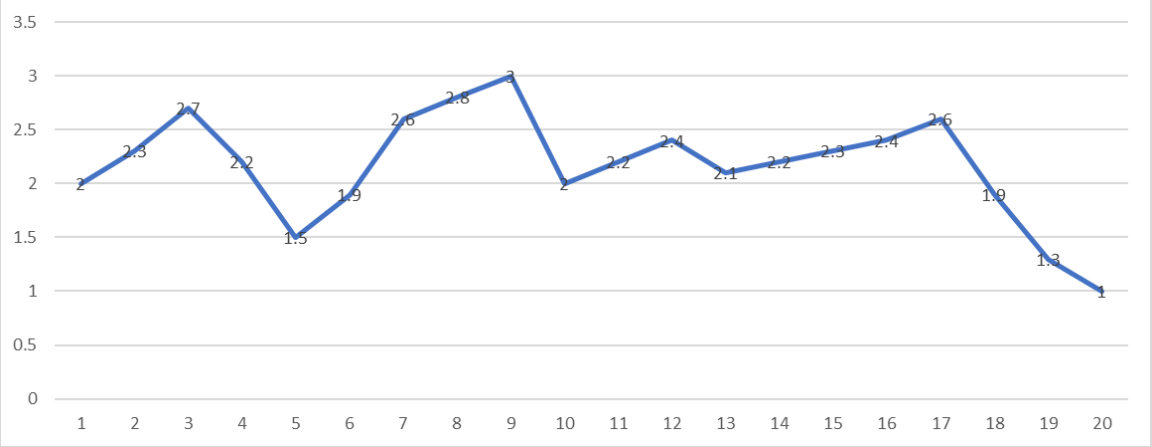


Рисунок 2.1 – Класичний графік адаптивного алгоритму підбору завдань

Проте недоліком даного підходу є відкрите питання визначення моменту завершення тестування і оцінки користувача. Даний алгоритм не вирішує те, коли тестування завершено і результат може бути представлений. В переважній більшості тестові набори програмуються із заздалегідь встановленою кількістю завдань, наприклад два користувачі отримують по 20 завдань. Будь-яке адаптивне тестування розпочинається із завдань «середньої» складності. Далі, в залежності від відповідей, наступні завдання пропонуються із підвищенням або із зниженням складності [3]. Якщо один із користувачів має дуже високий рівень знань – він все одно буде вимушений проходити 20 завдань, кожне наступне буде з яких буде пропонуватись складнішим ніж попереднє, і, ймовірно, лише

дійшовши до 15-18 завдання, він почне вирішувати завдання тієї складності, яка відповідає його рівню підготовки. Отримуємо ситуацію, коли людина вирішує 15 завдань, які є для неї занадто простими, і цей тестований міг би набрати більшу кількість балів, якби отримував більш складні завдання з самого початку.

З іншого боку тестований із низьким рівнем підготовки буде вимушений проходити 20 завдань, кожне друге яких буде провалено. Тобто перше, просте завдання вирішено успішно, але наступне, вже більшої складності, буде провалено, далі знову просте питання – вирішене і наступне, складніше, провалене. Даний тестований втратить половину від можливих балів через те, що йому будуть пропонуватись завдання, які він не може вирішити.

Окрема проблема, яку варто підняти – це час проходження тестування. В описаних вище сценаріях всі 20 питань призводять до прогнозованого результату, проте людина витрачає час даремно на повторення однакового шаблону відповідей.

В запропонованій системі буде підготовлено класифікаційну модель машинного навчання, яка буде приймати рішення коли тестування завершене, та яку оцінку отримує тестований. Кожне наступне вирішення завдання модель буде намагатись класифікувати користувача відповідно до відповідей на його попередні завдання. Класи будуть представлені оцінками [5, 7]. Приклад такої класифікації зображено на рисунках 2.2, 2.3.

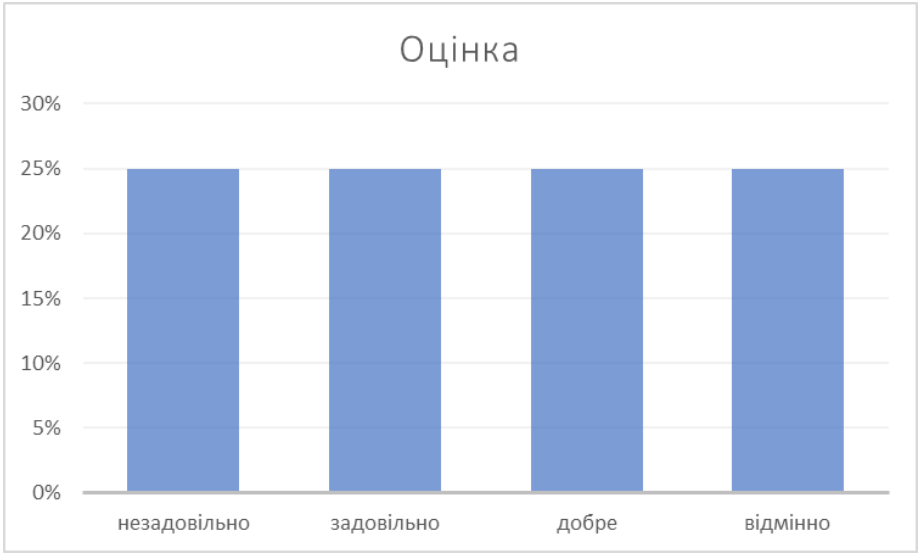


Рисунок 2.2 – Класифікація оцінки до початку тестування

Класи «незадовільно», «задовільно», «добре», «відмінно» наведені для спрощення. В розроблюваній моделі класи будуть представлені числом від 1 до 100, що виражає відсоток від максимально можливої кількості балів. Це допомагає моделі бути «не прив’язаною» до певної системи оцінювання, наприклад 5-бальної чи 12-бальної, і т.д. Це забезпечує працездатність моделі та звільняє від необхідності її перенавчати відповідно до нових класів (які різні в залежності від системи оцінок).

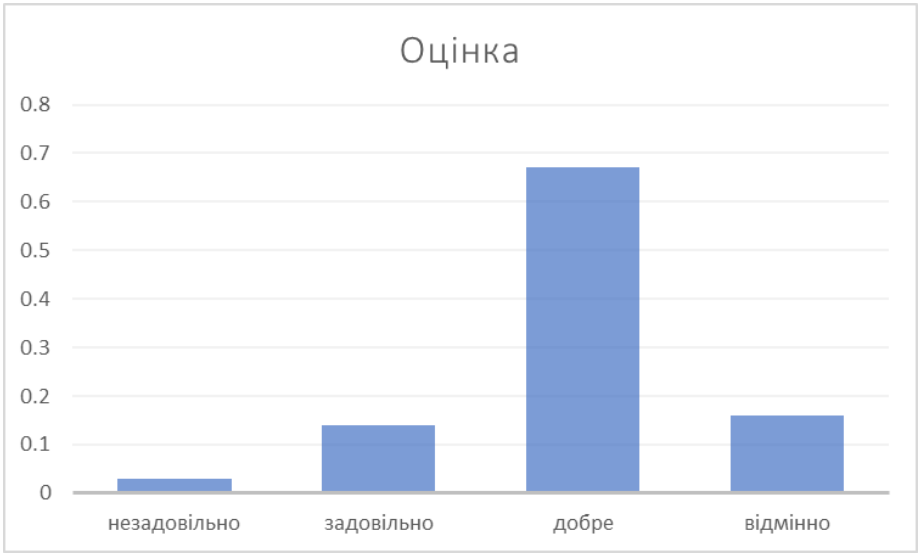


Рисунок 2.3 – Класифікація оцінки після проходження N завдання

Як тільки модель може класифікувати оцінку користувача, дійшовши певного зконфігурованого порогу (наприклад 70%), тестування автоматично завершується. Таким чином, система визначає шаблон відповідей на завдання, відповідно до їх складності та успішності відповіді. В такому разі може бути достатньо, наприклад, 10 з 20 завдань, щоб визначити, що користувач заслуговує оцінку відмінно. Іншому ж користувачеві потрібно буде всього 5 завдань щоб отримати оцінку незадовільно.

Важливо відзначити, що в такому підході неправильна відповідь зовсім не означає що оцінку «відмінно» вже не можна отримати. Оскільки користувачеві буде запропоновано дуже складні завдання, менш складні, прості, дуже прості і т. д. Як тільки система визначить, що користувач з 80% ймовірності

класифікується з оцінкою відмінно, тестування автоматично завершується [13].

Ключову роль в адекватності такого тестування грає якість моделі машинного навчання та адекватність її навчання, тому дану систему потрібно ретельно тестувати та підганяти під реальні вимоги [5, 10].

Важливим аспектом, який виконує дана модель – є визначення оцінки користувача. Варто зазначити що ця оцінка прямо залежить від навальних даних, та повинна бути визначена в них не як відношення суми правильних відповідей користувача до загальної суми завдань, запропонованих йому. В такому разі користувач зі слабким рівнем знань, який вирішив успішно багато простих завдань отримав би результат близький до 90%, так само, як і користувач з високим рівнем знань, який вирішив багато важких завдань.

Тому результат в навчальних вибірках повинен враховувати глобальний максимум, котрий може бути запропонований користувачеві, але враховуючи те, що кожен користувач починає тестування з середнього рівня складності та поступово його може збільшувати.

2.2 Вдосконалення методу «адаптивності» комп'ютерного тестування знань

Переважна більшість систем адаптивного тестування знань пропонують різноманітні типи завдань в залежності від їх складності індивідуально для кожного користувача. Але складність в свою чергу є числовою величиною, яку встановлює укладач тестового набору. Ця характеристика є цілком суб'єктивною і саме через вплив людського фактору, достовірність результатів тестування ставиться під сумнів. Часто складається ситуація коли найважчі завдання, за думкою розробника тесту, насправді мають середню складність, або навпаки.

Існуючі адаптивні системи тестування знань не пропонують ефективних механізмів коригування складності завдань. Тобто, запрограмована складність укладачем тесту, не змінюється і завжди залишається сталою величиною,

обраною повністю емпірично. Цей факт не дозволяє говорити про адекватність оцінки знань такою системою [1, 2].

Інша проблема, яка чітко виражена в існуючих відомих системах адаптивного тестування знань – інертність в підборі завдань, зумовлена жорстким коефіцієнтом зміна складності завдань. Вже нікого не дивує той факт, що користувач отримує важчі або легші завдання по мірі того, наскільки успішно вирішує попередні. На рисунку 2.4 наведено приблизний графік складності завдань, які пропонуються користувачеві у розробках аналогічних систем.

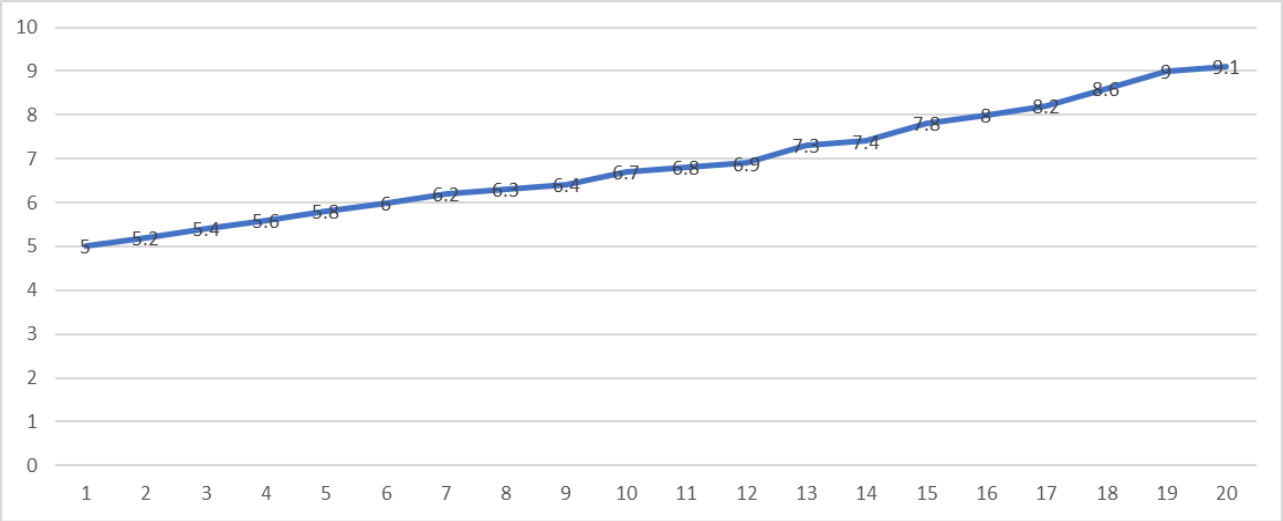


Рисунок 2.4 – Класичний графік «адаптивності» системи-аналога для користувача з високим рівнем підготовки

Припускаючи, що в цій системі користувач починає тестування із «середньої» складності та поступово отримує все важчі, або легші завдання відповідно до успішності його тестування, можемо бачити недолік системи. Такий користувач не має змоги розкрити свій потенціал і отримує «підходящі» по складності питання лише під кінець процесу тестування. Відповідно велику кількість балів буде втрачено, оскільки система в міру своєї «інертності» витрачає час і відведений ліміт питань для підлаштування своєї складності. В даному випадку користувач міг би успішно вирішити 20 завдань із складністю «10», але зміг отримати завдання складності «9» та більше лише під кінець процесу тестування. Аналогічна проблема існує для користувача із низьким

рівнем підготовки. На рисунку 2.5 наведено графік «адаптивності» системи-аналога для такого сценарію тестування.

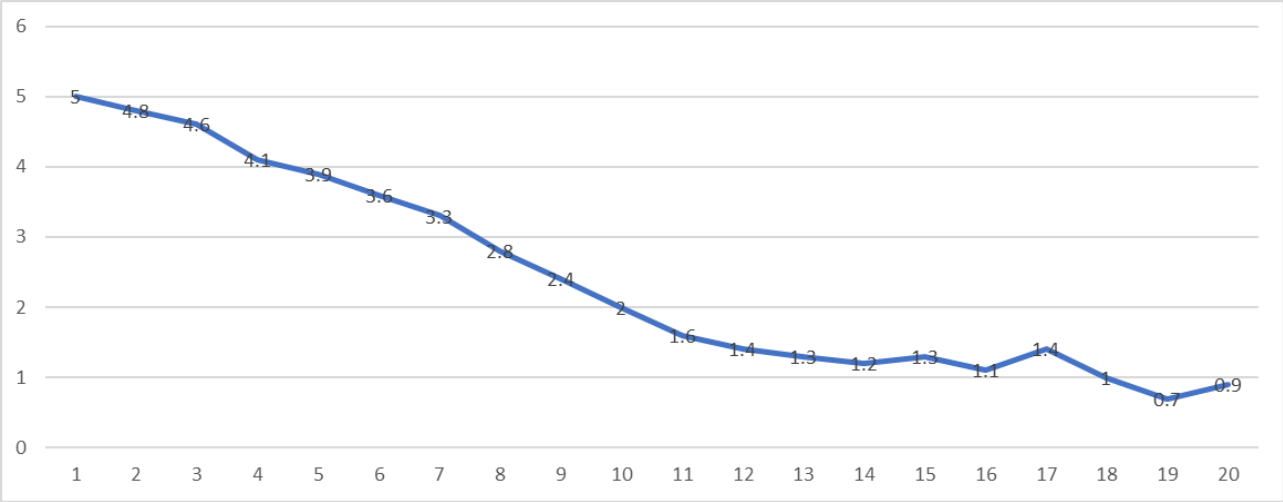


Рисунок 2.5 – Класичний графік «адаптивності» системи-аналога для користувача з низьким рівнем підготовки

Як добре видно з рисунку, користувач провалює всі завдання, аж до 15, оскільки система пропонує заважкі завдання для його рівня. В результаті користувач зміг успішно вирішити всього 3 із 20 завдань, і його результат тестування буде близький до 0. Тобто система не в змозі адекватно оцінити такого користувача, оскільки він не зміг показати свій рівень знань.

Пропонується вдосконалити метод «адаптивності» системи тестування знань таким чином, щоб вона ефективніше приймала рішення про вибір завдання для користувача. Для розв'язання цієї задачі пропонується розробити метод для нелінійного перерахунку складності завдань, щоб система якомога швидше підлаштовувалась під рівень знань користувача. В загальному, даний метод буде мати різкі «стрибки» по складності завдань на початкових завданнях, я на основі відповідей на декілька абсолютно різних по складності завдань система швидко вийде на «комфортну» складність для користувача [14].

Таким чином швидко вибудується середня лінія складності, по якій модель штучного інтелекту зможе точно класифікувати результат користувача, та зупинити тестування навіть раніше, ніж того потребують системи аналогів. Тобто користувачу не потрібно буде вирішувати всі 20 завдань. Як тільки

класифікаційна модель зможе передбачити результат процесу тестування, цей процес завершиться. Для вирішення поставленого завдання розроблено алгоритм обрахунку який виражається формулою 2.1. Графік з результатами роботи даного алгоритму «підлаштування» складності завдань наведено на рисунку 2.6.

$$D_i = \frac{M}{i-1} \cdot k_{i-1} + \frac{D_{i-1}+S}{2} \quad (2.1)$$

де D_i – складність завдання, запропонованого на i – ітерації тестування;

M – поточний результат користувача (сума всіх здобутих балів);

k – коефіцієнт прогресу, зменшується з кожним наступним завданням (0.8, 0.48, 0,36, 0.29…), розраховується за формулою 2.2;

S – успішність користувача, розраховується за формулою 2.3.

$$k_i = 1 - \frac{1+(k_{i-1})^2}{2} \quad (2.2)$$

де k_i – коефіцієнт прогресу на i – ітерації тестування.

$$S = \frac{M}{M_{max}} \quad (2.3)$$

Де M – поточний результат користувача (сума всіх здобутих балів);

M_{max} – максимально можливий результат (у випадку всіх правильних відповідей).

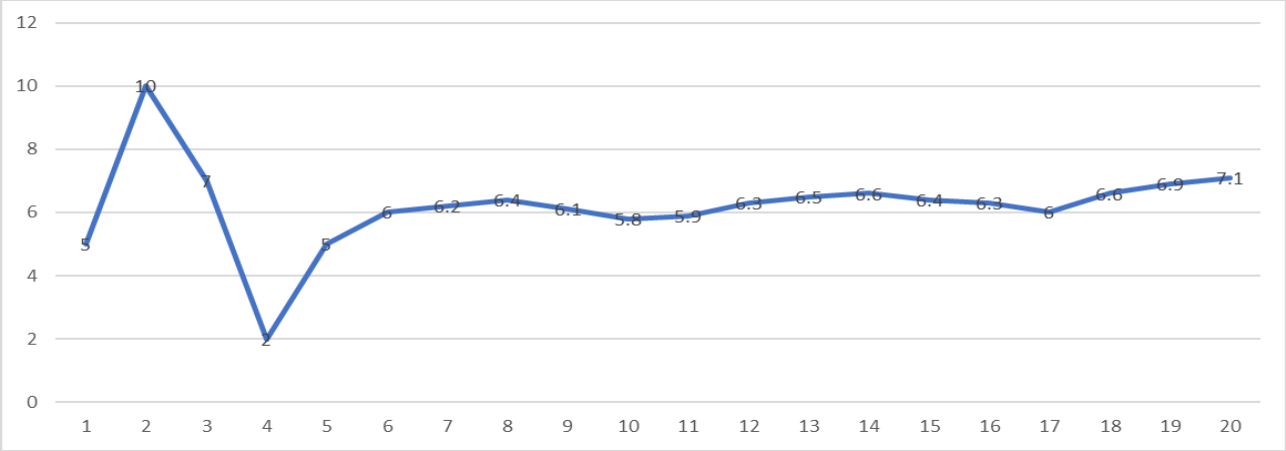


Рисунок 2.6 – Метод покращення «адаптивності» системи тестування знань.

Інше призначення даного методу – це коригування складності самих завдань. Якщо тестований з достатньо високим рівнем знань, не може вирішити

правильно завдання, яке по рівню складності відповідає його поточному рейтингу, то, ймовірно це завдання повинно мати більший рівень складності, ніж воно має. Звісно причина провалу може бути й у самому користувачеві, тобто його рівень знань не такий високий, як система його сприймає. Але в узагальненій вибірці поміж багатьох користувачів, наприклад, якщо 40 користувачів з рівнем «5» провалили завдання зі складність «5», а 15 користувачів з рівнем «5» успішно вирішили це завдання (тобто лише 23% користувачів впорались із завданням), то можна зробити припущення, що це завдання насправді не відповідає складності «5», а повинно мати складність, наприклад «5.4». Аналогічне припущення можна зробити і про завдання, які більшість користувачів вирішують успішно – найімовірніше вони мають завищену складність [15]. Запропонований метод повинен коригувати складність таких завдань на мінімальну величину, яка б не була помітною пересічному користувачу тесту. Якщо, після кожного вирішення або провалу завдання, його складність «підганяти» на дуже малу величину, то з часом це завдання досягне такого рівня складності, коли приблизно 50% користувачів будуть його вирішувати правильно, а 50% – неправильно. Саме таке значення можна вважати ідеально-відкаліброваним і відповідь на таке завдання може чітко визначити чи користувач досяг рівня цього завдання, чи він потребує завдання нижчого/вищого рівня [16]. Блок-схему розробленого методу корекції завдань зображено на рисунку 2.7.

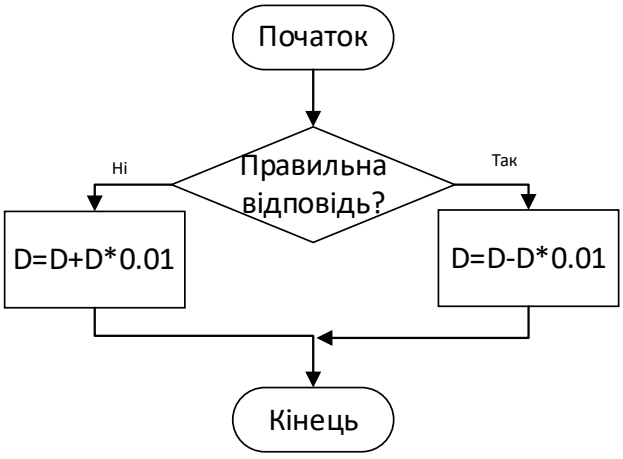


Рисунок 2.7 – Метод корекції завдань

Звичайно, що один користувач системи не зможе об’єктивно встановити складність всім завданням, але у довгостроковій перспективі, статистичні дані відповідей кожного користувача відкоригують складність завдань максимально правильно [4].

2.3 Проектування програмного засобу для адаптивного тестування знань

Для розробки програмного забезпечення для адаптивного тестування знань обрано клієнт-серверну архітектуру. Вся бізнес-логіка буде виконуватись на серверній частині. Клієнтський додаток комунікує з серверним через мережу Інтернет. Система включає окремий сервер бази даних.

Узагальнену діаграму компонентів подано на рисунку 2.8.

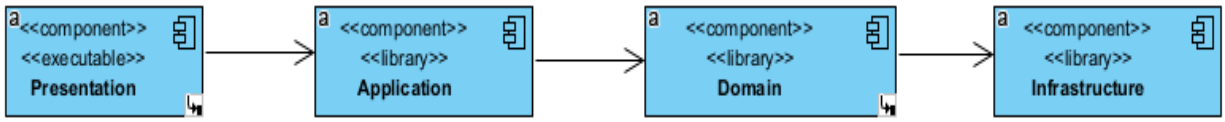


Рисунок 2.8 – Узагальнена діаграма компонентів системи

Серверний додаток містить наступні компоненти, які побудовані у вигляді окремих бібліотек класів:

- application – бібліотека класів для координації запитів від клієнтського додатку до класів сутностей для виконання бізнес-сценаріїв;
- domain – класи сутності, які інкорпорують бізнес-логіку;
- infrastructure – класи для роботи з відокремленою базою даних.

Клієнтський додаток містить набір класів для обробки дій користувача та комунікації з серверним додатком.

База даних працює на віддаленому сервері, а також може використовуватись будь-яке хмарне рішення для надійного і швидкого доступу до даних.

Проектування системи було виконано мовою UML, оскільки це

найвідоміший стандарт для моделювання і проектування. Мова UML має величезну аудиторію користувачів та розробників, тому документації по її використанню є безліч, що значно спрощує роботу та вирішення будь-яких складнощів під час проектування системи [17].

Найпершим етапом проектування був аналіз функціональних вимог системи та моделювання варіантів використання за допомогою діаграми прецедентів. Діаграму прецедентів зображено на рисунку 2.9.

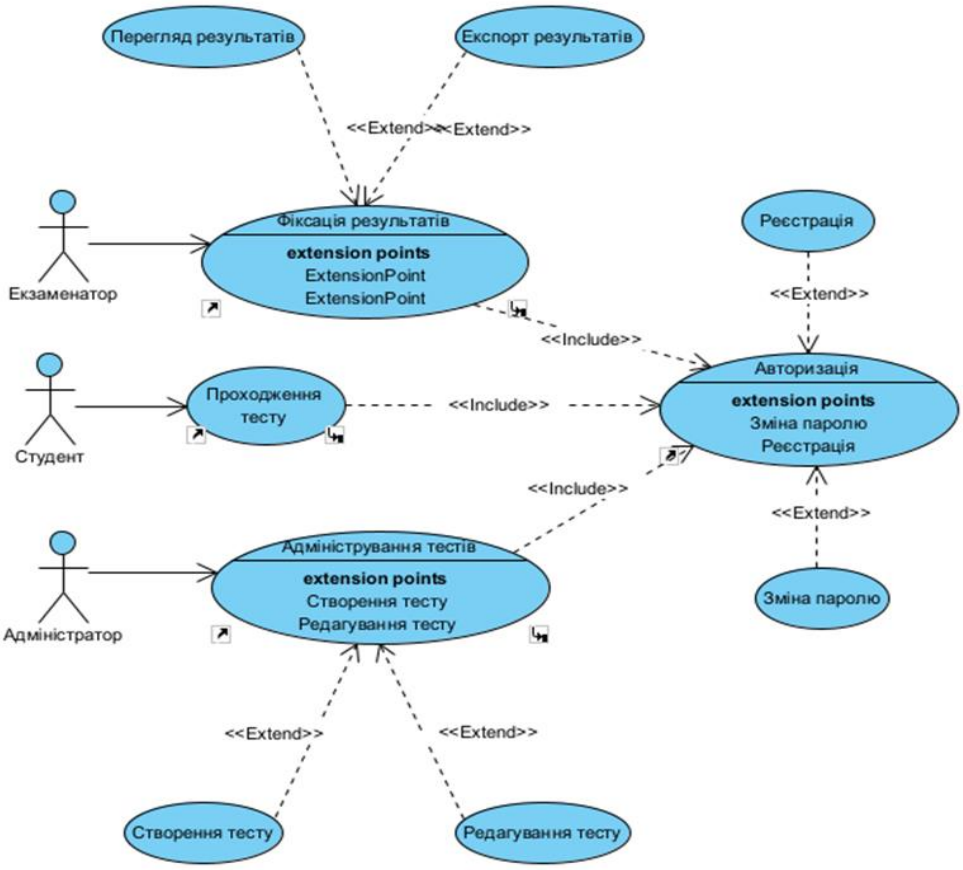


Рисунок 2.9 – Діаграма прецедентів

На цьому етапі важливо розуміти що система повинна буде робити, а не те, як вона це буде робити.

На діаграмі прецедентів зображено всіх можливих акторів системи:

- тестований – проходить тести;
- екзаменатор – перевіряє результати тестованих;
- адміністратор – створює та редагує тестові завдання.

Опис прецедентів:

- проходження тесту – єдиний сценарій, доступний для студента, дозволяє пройти тест, і переглянути результат;
- перегляд результатів – сценарій для екзаменатора, дозволяє переглянути результат тестування по певному студенту, або групі студентів;
- експорт результатів – сценарій для екзаменатора, дозволяє скачати CSV файл з результатами тестування по певному студенту, або групі студентів;
- створення тесту – сценарій для адміністратора, дозволяє створювати тестові набори, додавати до них завдання, встановлювати правильні відповіді, базову складність, додавати картинки;
- редагування тесту – сценарій для адміністратора, дозволяє редагувати існуючий тестовий набір;
- реєстрація – сценарій, доступний для всіх акторів, дозволяє зареєструватись в системі;
- авторизація – сценарій, доступний для всіх акторів, дозволяє авторизуватись в системі;
- зміна паролю – сценарій, доступний для всіх акторів, дозволяє змінити пароль.

Наступний етапом проектування є побудова діаграми класів. Розглядається вона з концептуальної точки зору, абстрагуючись від програмної реалізації і мови програмування. Класи на діаграмі групуються за призначенням і логічною схожістю у пакети. Діаграму класів зображено в додатку В.

Діаграма класів зображує статичні зв’язки між класами. Для кращого розуміння операцій та процесів в системі використовують діаграми послідовності. Така діаграма дає розуміння послідовність та характер зв’язків між класами [18]. Діаграму послідовності для сценарію використання системи «Проходження тесту» актора «Студент» зображено на рисунку 2.10.

На діаграмі зображено повний цикл бізнес-сценарію «проходження тесту» з послідовністю взаємодій всіх об’єктів системи. Наглядно зрозуміло напрямки викликів та їх характер. Також зображено основні логічні структури, які

відповідають основним логічним операціям всіх мов програмування: цикли, умовні операції, рекурсивні виклики.

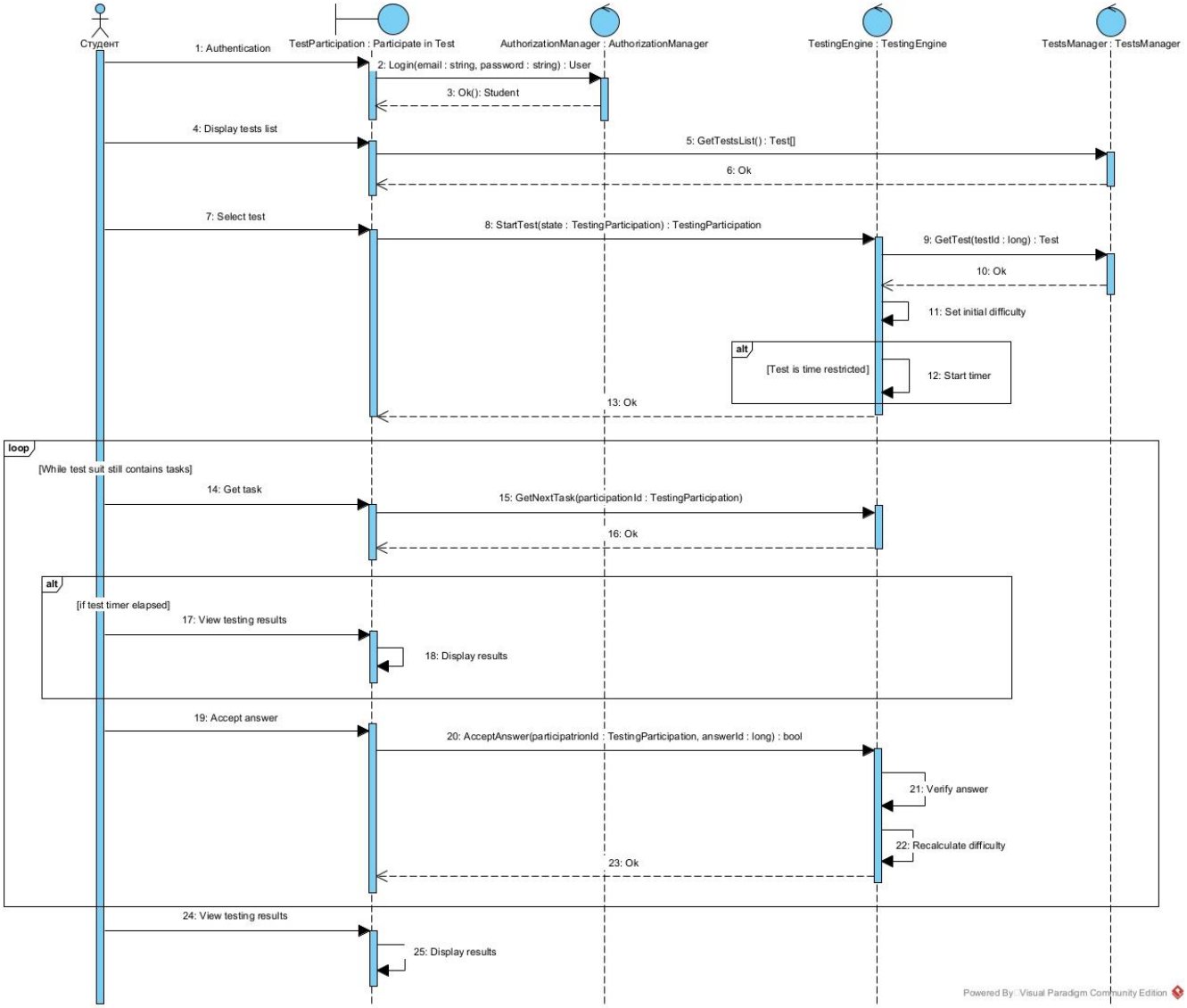


Рисунок 2.10 – Діаграма послідовності для прецеденту «Проходження тесту»

Окремим випадком діаграми послідовності є діаграма взаємодії. Вона концентрує увагу на самих зв’язках, їх типах, напрямку, а не послідовності.

Visual Paradigm дозволяє створювати діаграми взаємодії автоматично на основі діаграм класів і навпаки.

Хоча обидва типи діаграм моделюють взаємодію об’єктів і послідовність викликів, діаграма взаємодії надає більш високорівневий вигляд взаємодій. В свою чергу діаграма послідовностей моделює час життя об’єктів і часовий проміжок в який певна комунікація між об’єктами відбувається [19].

Діаграму взаємодії для прецеденту «Проходження тесту» зображено на рисунку 2.11.

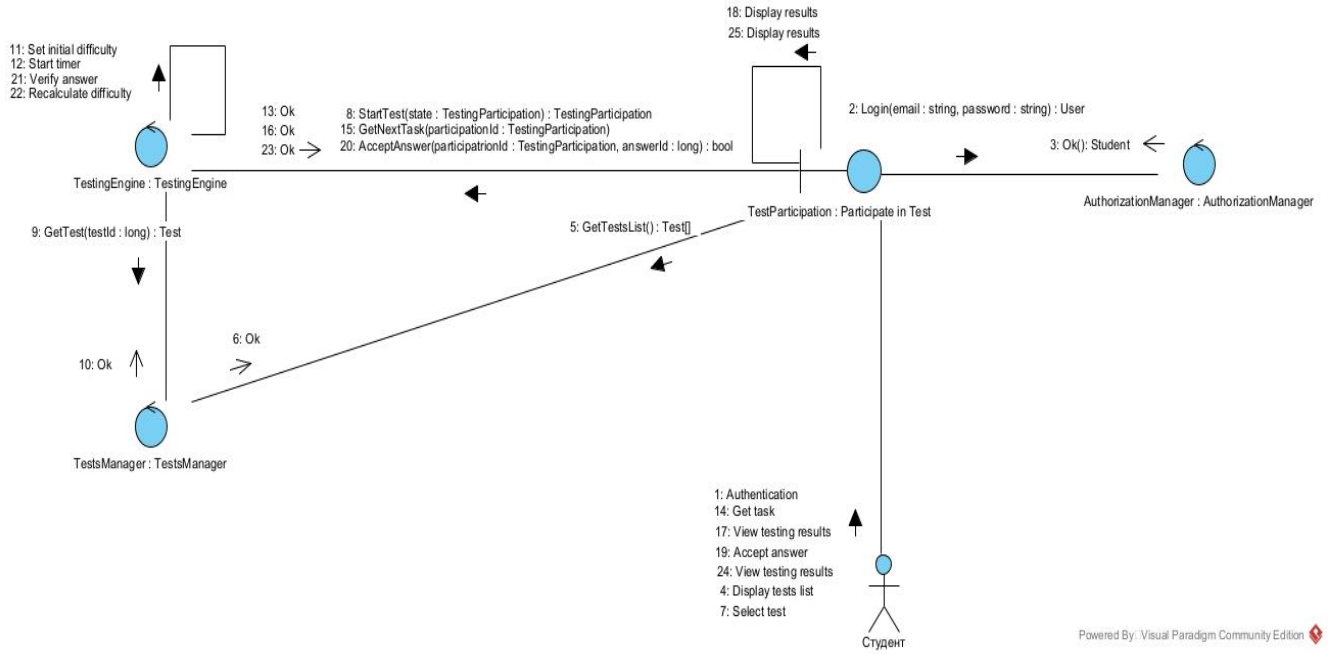


Рисунок 2.11 – Діаграма взаємодії для прецеденту «Проходження тесту»

Наступний етап проектування – створення діаграм діяльності. Дані діаграми використовуються для моделювання послідовності дій (реалізованих методами класів), або процесів в системі. Вони дозволяє візуалізувати, аналізувати та моделювати бізнес-процеси або алгоритми. Ключові елементи та поняття таких діаграм:

- дія – представляє окрему дію або обчислення в процесі. Наприклад, "створити обліковий запис" або "вивести результат";

- вузол – вказує на виконання певної дії або обчислення. Може представляти вхідні або вихідні дані;

- керуючий потік – лінії, які вказують порядок виконання дій або переходи між ними

Діаграму діяльності для прецеденту «Проходження тесту» наведено на рисунку 2.12. Дана діаграма описує загальну бізнес-логіку процесу тестування від моменту відкривання додатку до моменту його закривання.

Діаграми діяльності допомагають людям із відділу бізнесу та технічним спеціалістам узгодити всі деталі процесів та поведінки системи. Вони

відображають покроково послідовність взаємодії користувачів із системою. Кожна вершина на діаграмі відповідає за певну дію або діяльність, а переходи між вершинами – за напрямок потоку виконання сценарію в системі. Напрямок може змінюватись в залежності від виконання певних умовних операторів.

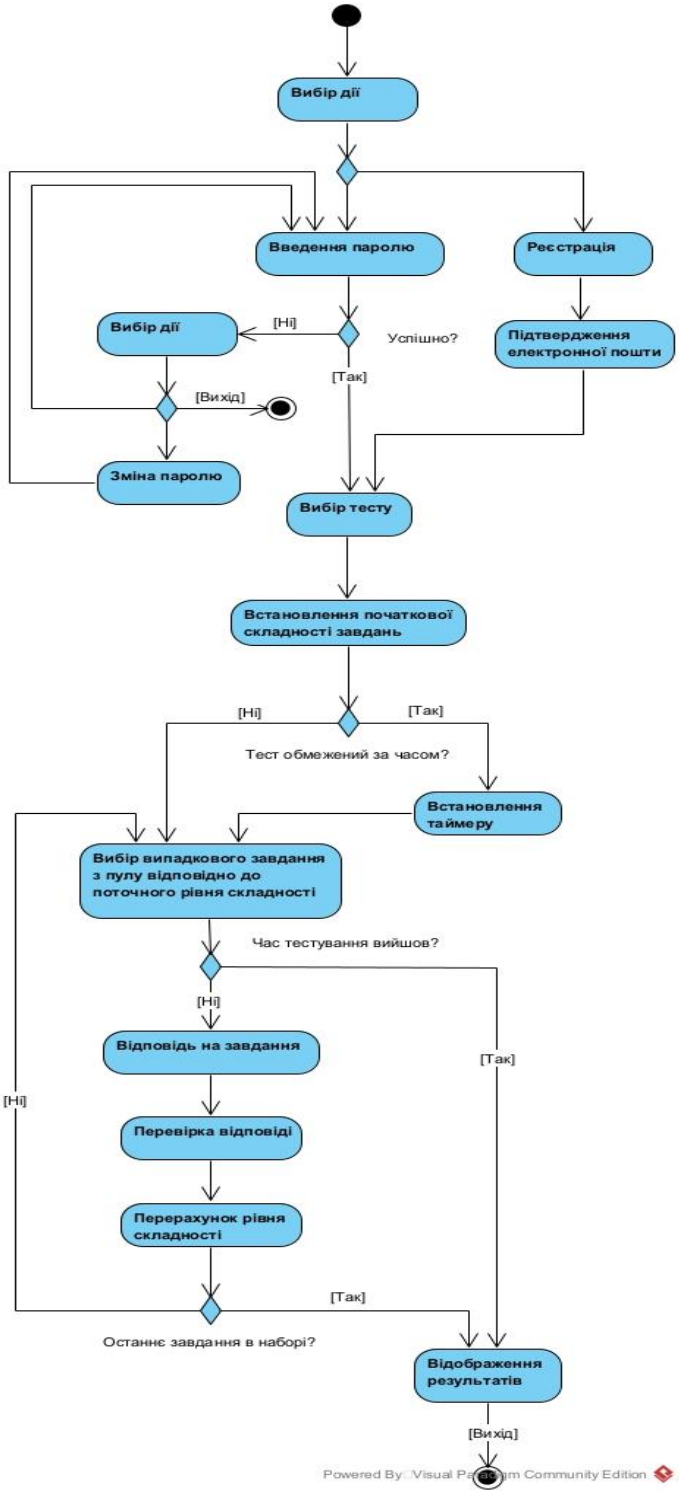


Рисунок 2.12 – Діаграма діяльності для прецеденту «Проходження тесту»

Однією із варіацій діаграм діяльності є Swimlane діаграми. Такі діаграми

показують взаємодію між акторами системи в межах спільного бізнес процесу. Swimlane діаграму діяльності для всіх акторів наведено на рисунку 2.13.

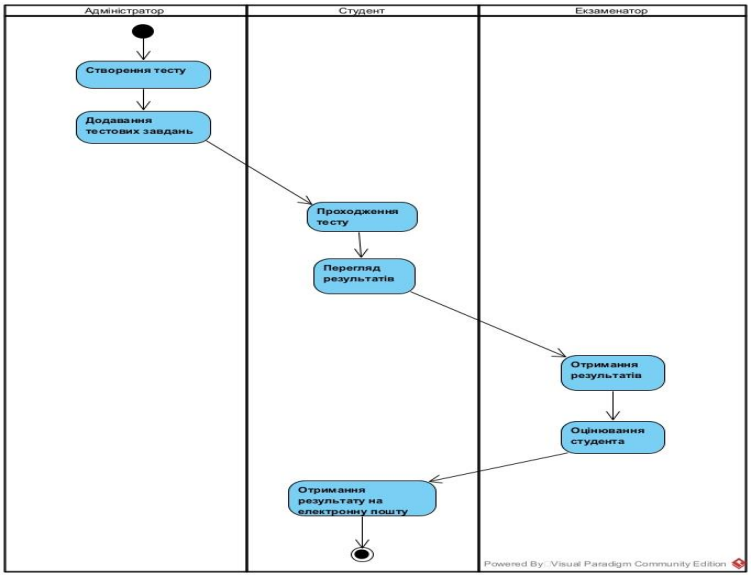


Рисунок 2.13 – Swimlane діаграма діяльності

Крім діаграм діяльності, використовують також діаграми станів. Такі діаграми описують не дії, які відбуваються в системі, а стани об’єктів, які набуваються в результаті певних дій. Діаграму станів для об’єкту «Студент» зображено на рисунку 2.14.

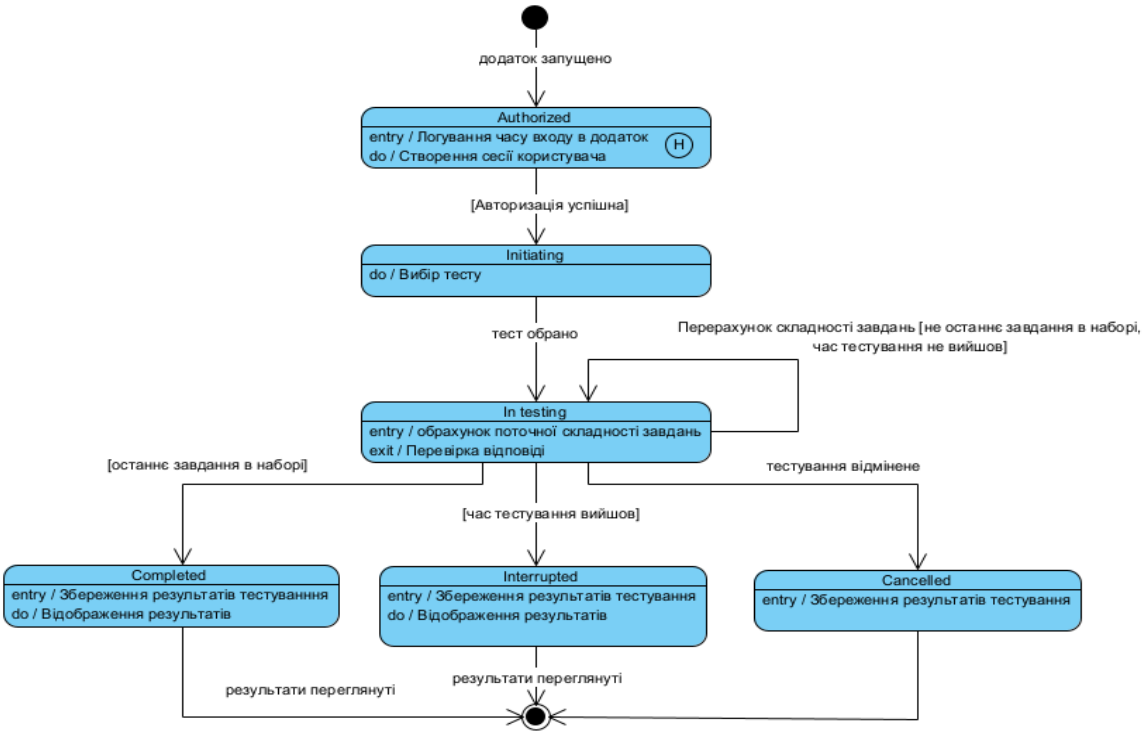


Рисунок 2.14 - Діаграма станів для об’єкту «Студент»

Дана діаграма описує стан об’єкту Student на різних етапах тестування. Кожна вершина графу – це певний статус об’єкту системи, а переходи між вершинами – це дії, або умови, які виконуються для встановлення цього статусу. Діаграмам станів притаманні розгалуження (умовні оператори) та сходження потоків виконання [17-20]. Як тільки користувач авторизується в системі, його статус встановлюється як «Authorized». Далі користувач вибирає один з доступних тестових наборів. При цьому його статус «Initiating». Після вибору тесту, статус змінюється на «In testing». Далі користувач має три варіанти зміни статусу: Completed – при нормальному проходженні всіх тестових завдань, Interrupted – при закінченні часу тестування, Cancelled – при відмові від проходження тестування.

2.4 Висновки

Запропоновано шляхи вдосконалення методів та алгоритмів проведення адаптивного тестування знань. Проведений аналіз показав, що задача класифікації користувачів користувачів за рейтингом і тестових завдань за складністю може бути ефективно вирішена засобами машинного навчання. Складність при впровадженні такого підходу може виникати при виборі алгоритму моделі та підготовці даних для навчальної вибірки.

Таким чином, використання технологій машинного навчання може покращити якість та швидкість процесу адаптивного тестування знань. Це, в свою чергу, зекономить ресурси та витрати на організацію таких заходів.

3 ПРОЕКТУВАННЯ ТА НАВЧАННЯ МОДЕЛІ ШТУЧНОГО ІНТЕЛЕКТУ

3.1 Постановка задачі для вирішення засобами машинного навчання

Аналіз даних як наука і галузь дослідження на сьогоднішній день розвивається величезними темпами. Це дає змогу делегувати важкі, рутинні процеси на модель, яка собою представляє функцію, котра може приймати рішення над даними, які їй передають. Якість таких рішень критично залежить від якості навчання моделі і потребує ретельного збору, підготовки та нормалізації.

Аналіз даних на підприємстві може включати в себе також збір сирих статистичних даних, які в подальшому можуть бути використані для навчання моделі та використання її на цьому ж підприємстві.

Приклади задач класифікації включають розпізнавання образів (наприклад, розпізнавання цифр на зображеннях), фільтрацію спаму в електронній пошті, визначення типу хвороби на основі медичних зображень, тощо. Ключовим елементом вирішення задачі класифікації є вибір і оптимізація моделі, використання відповідних функцій втрат та оцінок, а також ефективна обробка та аналіз даних [21].

Найважливішою частиною розроблюваної системи адаптивного тестування знань є модель штучного інтелекту, котра буде приймати рішення про те, коли тестування можна вважати завершеним. Саме цей модуль забезпечить підвищення швидкості процесу тестування та заощадження ресурсів, необхідних для проведення тестування.

Задача, яка ставиться в даній роботі зводиться до задачі класифікації. Класифікація в машинному навчанні - це процес призначення об'єкта (або елемента) до одного з попередньо визначених класів або категорій на основі його властивостей чи характеристик. Це один з основних видів завдань навчання з наглядачем.

Для вирішення цієї задачі необхідно виконати наступні кроки:

1. Підготовка даних: Збір і підготовка набору даних, який складається з атрибутів вхідних об'єктів і відповідних їм міток класів.

2. Вибір моделі: Вибір моделі для класифікації. Це може бути, наприклад, лінійна регресія, метод опорних векторів, нейронна мережа, дерево рішень чи інша модель машинного навчання.

3. Навчання моделі: Модель навчається на тренувальних даних, де вона "вивчає" відповідності між вхідними об'єктами і їх класами.

4. Оцінка моделі: Після навчання моделі важливо оцінити її ефективність на тестових даних, щоб переконатися в її здатності узагальнювати вивчені залежності і оцінювати з їх допомогою нові дані.

5. Прогнозування нових даних: За допомогою навченої моделі можна класифікувати нові, раніше невідомі дані [22].

3.2 Вибір програмних засобів для підготовки моделі

Для дослідження поставленої задачі та підготовки моделі було обрано пакет програмного забезпечення WEKA (Waikato Environment for Knowledge Analysis), оскільки він він гарно себе зарекомендував за останнє десятиліття в області аналізу даних та машинного навчання, має широкий функціонал, доступну документацію в мережі Інтернет, а також є безкоштовним.

Це популярна відкрита платформа для розробки та використання алгоритмів машинного навчання. Вона написана на Java і надає інтуїтивно зрозумілий графічний інтерфейс, а також Java API для програмістів. Weka має багато можливостей, які полегшують використання та експериментування з алгоритмами машинного навчання. Основні можливості Weka включають:

- має широкий вибір алгоритмів машинного навчання, які охоплюють класифікацію, кластеризацію, регресію, асоціативне навчання та інші завдання;

- надає засоби для візуалізації даних, включаючи графіки, діаграми та інші інструменти, які допомагають аналізувати дані перед застосуванням алгоритмів машинного навчання;

- має функціонал для попередньої обробки даних, включаючи видалення відсутніх значень, видалення шуму, кодування категоріальних змінних та інші операції;

- може бути використана разом з іншими інструментами та мовами програмування, такими як Java. Її Java API дозволяє програмістам легко інтегрувати Weka в свої власні програми;

- має ряд інструментів для валідації моделей та оцінки їх продуктивності за допомогою різних метрик якості;

- підтримує як навчання з учителем, так і навчання без учителя, що дозволяє використовувати різні алгоритми для різноманітних завдань.

- дозволяє користувачам створювати та виконувати складні експерименти з використанням різних алгоритмів та параметрів.

- має активну спільноту користувачів, докладну документацію та навчальні матеріали, що полегшує вивчення та використання інструменту.

Робоче вікно програмного забезпечення WEKA зображено на рисунку 3.1.

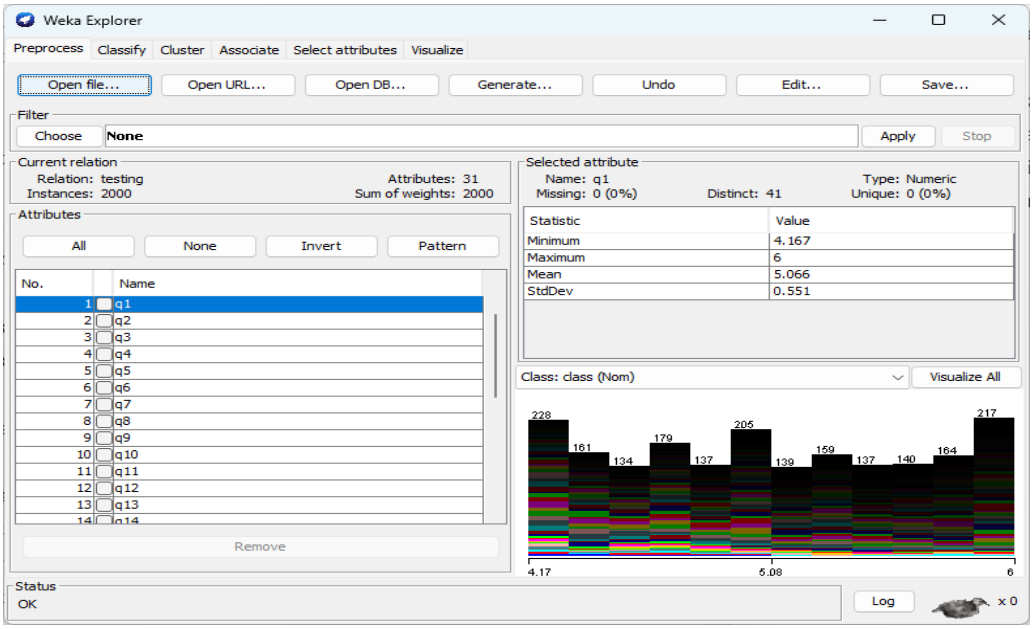


Рисунок 3.1 – Робоче вікно програми WEKA

Дане програмне забезпечення дозволяє виконати всі мінімально необхідні дії для підготовки моделі штучного інтелекту, готової до використання в адаптивній системі тестування знань [23].

3.3 Вибір типу та алгоритму навчання моделі

Існує багато типів машинного навчання. Найвідоміші з них це:

- навчання з учителем (Supervised Learning) – модель навчається на визначеному наборі даних, де для кожного вхідного прикладу є відомий результат. Задачі включають класифікацію (розподіл прикладів у задані категорії) та регресію (прогнозування числового вихідного значення);

- навчання без учителя (Unsupervised Learning) – модель навчається на непозначених даних, і алгоритм самостійно виявляє закономірності та структури в даних. Задачі включають кластеризацію (групування подібних прикладів) та асоціативне навчання (виявлення правил асоціацій між елементами);

- навчання з підкріпленням (Reinforcement Learning) – модель навчається приймати рішення, взаємодіючи з динамічним середовищем. Модель отримує «винагороду» або «покарання» за кожен варіант дії, і її мета - максимізувати винагороду протягом часу.

Крім описаних вище типів, існують також різноманітні їх комбінації, проте вони використовуються для дуже специфічних завдань. Для розробленого методу було обрано саме навчання з учителем, оскільки перед розробкою стоїть саме задача класифікації [24].

Вибір алгоритму навчання для моделі класифікації залежить від різних факторів, таких як характеристики даних, розмір набору даних, вимоги до часу виконання, вимоги до інтерпретованості моделі та інші конкретні вимоги проекту. Найбільш популярними алгоритмами класифікації є:

- логістична регресія – добре підходить для бінарної класифікації. Використовується, коли важлива інтерпретованість результатів та взаємозв'язки між атрибутами та ймовірностями класів;

- метод опорних векторів (SVM) – ефективний для бінарної і мультикласової класифікації. Добре працює в умовах високої розмірності або коли є невеликий об'єм даних;

- дерево рішень та випадковий ліс – відмінний вибір для обробки великих об'ємів даних. Дерева рішень використовуються для простої інтерпретації, тоді

як випадковий ліс забезпечує більшу стійкість до перенавчання;

- нейронні мережі – використовуються для складних завдань, таких як розпізнавання образів або обробка природних мов. Для великих об'ємів даних та завдань глибокого навчання;

- K-найближчих сусідів (k-NN) – використовується для простих завдань класифікації, особливо тих, де схожість між прикладами грає ключову роль.

- наївний Байєсівський класифікатор – часто використовується для текстового аналізу та завдань, де припущення про незалежність атрибутів може бути виправданим.

- градієнтний бустінг – ефективний для вдосконалення результатів слабших моделей. Часто використовується в конкурсах з машинного навчання.

- регресія – метод, який полягає в аналізі зв'язку між декількома незалежними змінними і залежною змінною;

- дискримінантний аналіз – метод що дозволяє оцінювати відмінності між двома і більше групами об'єктів по декільком змінним одночасно.

Вибір конкретного алгоритму також може включати етап налаштування гіперпараметрів для досягнення найкращих результатів на конкретному наборі даних. Важливо провести експерименти та враховувати контекст проекту для вибору найбільш підходящого алгоритму [25]. Для досліджуваної розробки було обрано декілька найбільш підходящих алгоритмів навчання: K-найближчих сусідів, наївний Байєсівський класифікатор та дерево рішень. Після навчання необхідно порівняти правильність роботи отриманих моделей для поставленої задачі та обрати найкращий результат.

Дерево рішень являє собою графічне представлення процесу послідовного прийняття рішень. Цей алгоритм описує вузли, в яких вирішуються умови, та гілки, які відходять від вузлів і слугують для прийняття альтернативних рішень. Кожна гілка має свою вагу, яка представляється її листком. Загальна очікувана корисність будь-якого завдання обчислюється як зважена сума усіх гілок від рішення до всіх листків, що належать даній гілці. Перевагою такої моделі є те, що вона показує кожне можливе рішення, попередні та наступні рішення для

нього, а також кінцеві наслідки від кожного рішення. Так, як листки являють представляють собою обмежені сутності, які містять лише одну ознаку, то вони і вказують на результат класифікації. Перевагою дерева рішень є те, що воно дозволяє визначити наскільки висока ймовірність результату класифікації. Ця величина обчислюється за допомогою формули Шенона 3.1 [26].

$$H(t,D) = -\sum[\ P(t = i) * log_2(P(t = i))\ ] \quad (3.1)$$

Де *P(t=i)* – ймовірність того, що шукана ознака *t* належить класу *i*;

l – різні класи цільової ознаки у наборі даних *D*.

Фактично дерево рішень містить явне перерахування можливих варіантів вибору, до яких алгоритм приходить щоразу коли класифікує набір вхідних даних.

Метод К-найближчих сусідів є одним з найпростіших серед алгоритмів класифікації. Його принцип полягає в обчисленні відстаней між невідомої вибіркою і всіма відомими вибірками. В результаті К відомих вибірок, котрі мають найменшу відстань до невідомої вибірки обираються як результат класифікації, тобто відносяться до однієї категорії з невідомою вибіркою. В загальному випадку алгоритм k-найближчих сусідів можна записати за допомогою формули 3.2.

$$a(x) = \text{argmax}\ y \in Y \sum[yi; x = y]\omega(i, x)\ m\ i{=}1 \quad (2.17)$$

де ω (i; x) - задана вагова функція, яка оцінює наскільки сильно і-й сусід впливає на об'єкт x.

Наївна модель Байєса – це простий імовірнісний класифікатор, в основі якого лежить застосування теореми Байєса зі строгими припущеннями (наївними) про незалежність. Наївні Байєсівські класифікатори можуть навчатись дуже ефективно і працювати набагато краще нейронних мереж у багатьох складних сценаріях використання. Перевагою даної моделі є мала

кількість вхідних даних, необхідних для навчання, оцінки параметрів і класифікації. Дана модель представляє вірогідну модель [27].

3.4 Підготовка даних для навчання моделі

Оскільки в даній системі рейтинг користувача, відповідно якого і підбираються завдання за складністю, змінюється в процесі тестування після кожного пройденого завдання, а кількість вирішених завдань і їх складність вирішує кінцеву оцінку користувача, то доцільно навчати модель саме на цих даних.

Під час тестування рейтинг користувача, та складність завдань, які йому пропонуються змінюється як зображено на рисунку 3.2.

Завдання №	1	2	3	4	5	6	7	8
Успішність вирішення завдання	невдало	невдало	невдало	невдало	успішно	успішно	успішно	успішно
Складність запропонованого завадння	5	4.16666667	3.894080997	3.245067497	2.950061361	2.950061361	3.392570565	3.697901916
Перерахований рейтинг користувача	5	4.95049505	3.858024691	3.780661162	2.750057201	2.979561975	3.097564429	3.460421977
Отримано балів	0	0	0	0	2.950061361	2.950061361	3.392570565	3.697901916
Завдання №	9	10	11	12	13	14	15	16
Успішність вирішення завдання	невдало	успішно	успішно	успішно	успішно	успішно	успішно	невдало
Складність запропонованого завадння	4.326545242	4.28370816	4.840590221	5.276243341	5.592817942	6.487668813	6.487668813	7.785202575
Перерахований рейтинг користувача	3.919776031	4.24171102	4.712078977	4.937402026	6.225967143	5.648746121	7.590572511	7.590572511
Отримано балів	0	4.28370816	4.840590221	5.276243341	5.592817942	6.487668813	6.487668813	0

Рисунок 3.2 – Робота алгоритму адаптивності

На вхід модель повинна приймати рейтинг користувача на кожному етапі тестування, і намагатись прогнозувати результуючу оцінку. Таким чином модель буде намагатись підібрати шаблони відповідей попередніх користувачів, які вирішували завдання з подібною успішністю і оцінити поточного користувача так само. Як тільки модель зможе з ймовірністю 70% класифікувати оцінку користувача, тестування можна вважати завершеним. Поріг 70% вибирається емпіричним шляхом на основі тестування системи і спостереження за оцінками користувачів.

Для навчання моделі використовується формат даних ARFF (Attribute-

Relation File Format), який визначає структуру та представлення даних для використання у задачах машинного навчання. ARFF-файл містить дві основні частини: опис атрибутів (attributes) і дані (data) [28].

Опис атрибутів (Attributes): Ця частина визначає назви і типи атрибутів у наборі даних. Кожен атрибут характеризує певну характеристику об'єкта. Приклад опису атрибутів зображено на рисунку 3.3.

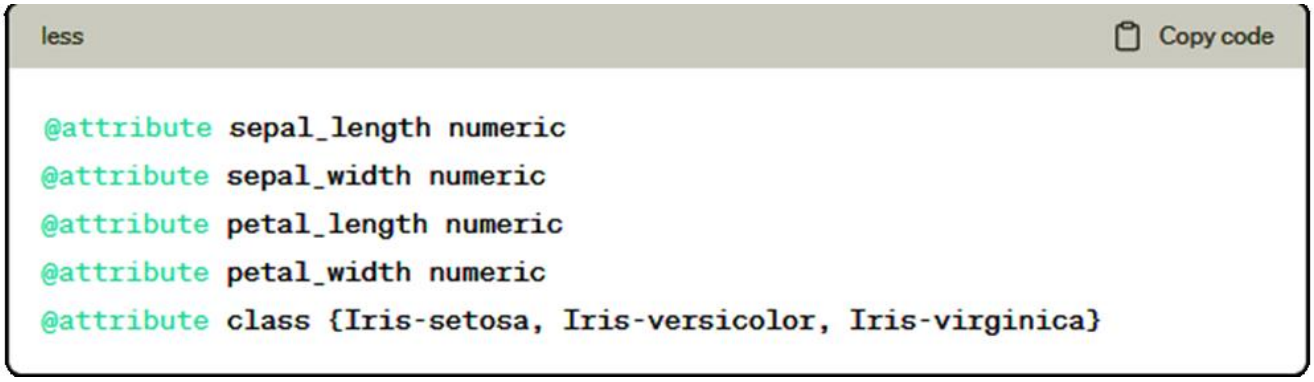


Рисунок 3.3 – Приклад опису атрибутів у файлі ARFF

В даному прикладі наведено чотири числових атрибути і один атрибут з визначеним списком можливих значень (class), що представляє собою клас об'єкта. Саме такий тип даних використовується для класифікації вхідних даних.

Опис даних (Data): Ця частина містить реальні значення для кожного об'єкта у наборі даних. Кожен рядок відповідає одному об'єкту, значення розділені комами [29]. Приклад опису даних зображено на рисунку 3.4.

Рисунок 3.4 – Приклад опису даних у файлі ARFF

В цьому прикладі перші чотири значення – це числові атрибути, а останнє

значення – це класовий атрибут.

Оскільки адаптивна система тестування знань перебуває на стадії розробки, то великої кількості статистичних даних про користувачів тестування не накопичено. Тому для навчання моделі набір даних в даній роботі готувався в середовищі MS Excel, за допомогою генерування великої кількості можливих варіантів проходження тестування.

Оскільки модель повинна навчатись на чітко визначеній кількості вхідних атрибутів, прийнято максимальну питань для одного сеансу тестування як 30.

Рейтинг користувача після кожного вирішеного завдання представлений як вхідний атрибут для моделі. А результат тестування (відсотковий від максимального) є класом об’єкту, який модель буде вчитись прогнозувати. За допомогою генерації випадкових сценаріїв тестування було підготовлено 1000 варіантів для навчального набору даних, та 1000 тестових наборів. Тестові дані необхідні для оцінки правильності роботи моделі після навчання. Фрагмент навчального набору приведено на рисунку 3.5.

```
@attribute q3 real
@attribute q4 real
@attribute q5 real
@attribute q6 real
@attribute q7 real
@attribute q8 real
@attribute q9 real
@attribute q10 real
@attribute q11 real
@attribute q12 real
@attribute q13 real
@attribute q14 real
@attribute q15 real
@attribute q16 real
@attribute q17 real
@attribute q18 real
@attribute q19 real
@attribute q20 real
@attribute q21 real
@attribute q22 real
@attribute q23 real
@attribute q24 real
@attribute q25 real
@attribute q26 real
@attribute q27 real
@attribute q28 real
@attribute q29 real
@attribute q30 real
%@attribute result real
%@attribute class {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20}
@attribute class
{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,
71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,97,98,99,100}

@data
4.807692308,4.807692308,4.040077569,4.000076801,3.603672794,3.028296466,2.588287577,2.310971051,1.958450043,1.688319003,1.6233
,1,1,1,1,1,1,1,0
5.9,5.784313725,5.211093446,4.737357679,4.083929033,3.679215345,3.285013701,2.959471803,2.690428912,2.42380983,2.071632334,1.8
19,1.30465271,1.087210592,1.055544264,1,1,1,1,1,1,1,1,2
5.2,5.928,5.488888889,5.178197065,5.126927787,4.381989562,4.213451502,3.795902254,3.721472798,3.352678196,2.915372344,2.858208
39109511,1.276234046,1.263598065,1.089308677,1,1,1,1,1,1,1,1,4
5.3,5.989,7.12691,6.251675439,5.897807018,5.084316394,4.58046522,4.404293481,3.829820418,3.791901404,3.791901404,3.240941371,3
9,1.272522766,1.272522766,1.189273613,1.061851441,1,1,1,1,1,1,1,6
5,5.9,7.08,7.08,6.156521739,6.156521739,5.648185082,5.043022394,4.757568297,4.13701591,4.096055357,3.864203167,3.825943729,3.3
2.277019312,1.946170353,1.818850797,1.699860558,1.666529959,1.602432652,1.526126336,1.293327403,1.220120192,1.09920738,8
```

Рисунок 3.5 – Фрагмент навчального набору даних у форматі ARFF

Після навчання моделі необхідно провести її тестування на тестових даних та прийняти рішення про її налаштування, перенавчання, або збереження для подальшого використання.

3.4 Навчання класифікаційної моделі

Для адаптивної системи тестування знань проводилось тренування моделей за допомогою алгоритмів дерева рішень, методу найближчих сусідів та наївного Байєсівського класифікатора. Процес навчання моделі за алгоритмом дерева рішень зображено на рисунку 3.6.

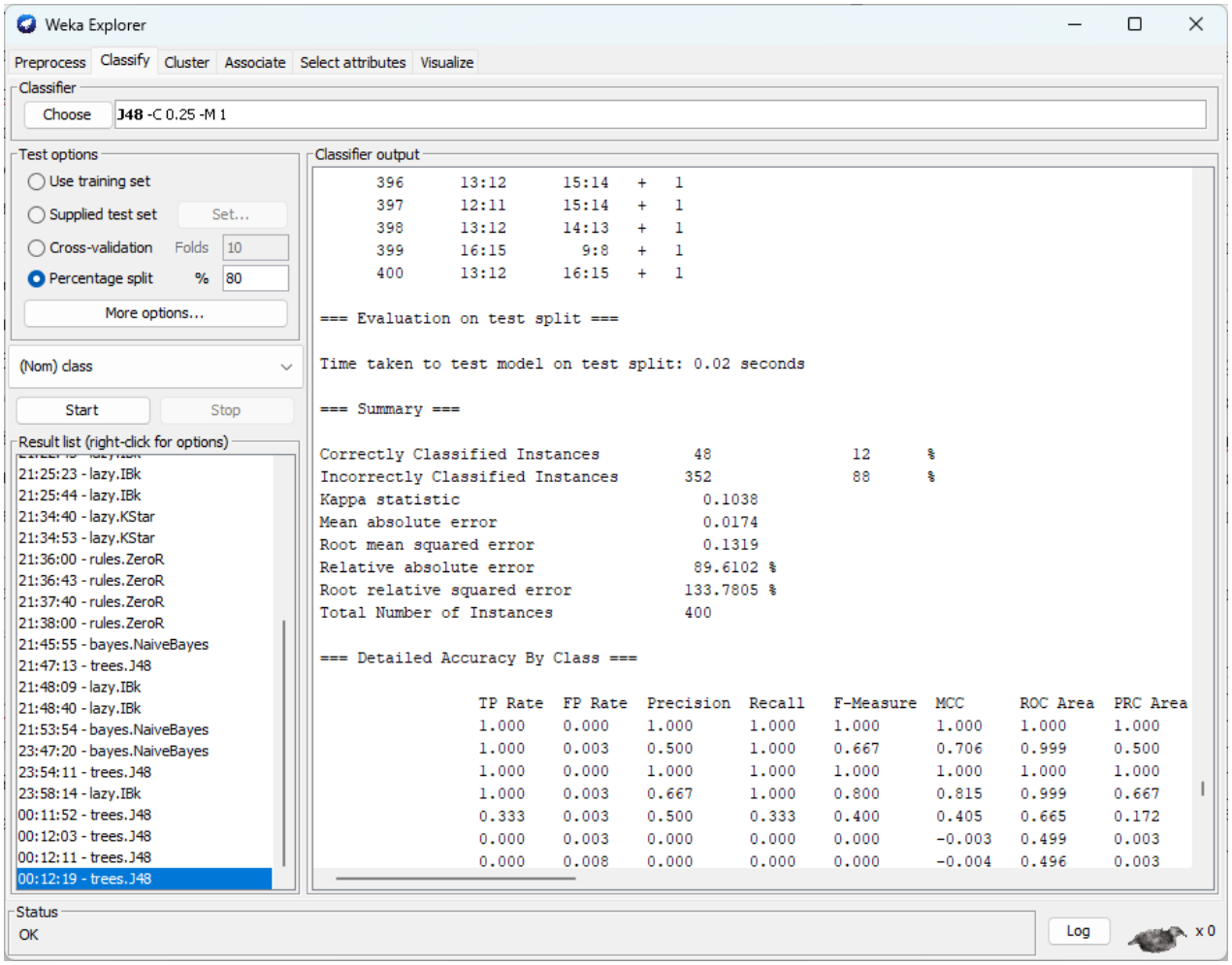


Рисунок 3.6 – Навчання моделі за алгоритмом дерева рішень

За допомогою Weka є можливість візуалізувати згенероване дерево рішень, щоб мати уявлення як вхідні дані проходять від початку дерева і до

кінцевого «листка» при класифікації цих даних. Варто зауважити що даний алгоритм вразливий до «перенавчання». Тобто якщо навчальна вибірка покриває занадто багато сценаріїв можливих вхідних даних, то модель втрачає можливість аналізувати невідомі їй дані, і працює лише із «завченими» наборами, що нівелює користь від її використання. Для запобігання такого явища використовують підхід, відомий як «обрізання гілок» (pruning).

Обрізання гілок у класифікаційних деревах — це техніка оптимізації, яка полягає в обмеженні розміру дерева за допомогою видалення частини його гілок. Мета обрізання – покращити загальну генералізацію моделі та запобігти перенавчанню. Це особливо важливо, коли дерево занадто глибоке та перетворюється в слабку узагальнюючу модель, яка добре пристосована до тренувальних даних, але погано справляється з новими даними.

Існують два основних методи обрізання гілок:

1. Pre-Pruning (обрізання перед побудовою) – цей метод включає обмеження росту дерева в процесі побудови. Дозволяє встановити максимальну глибину дерева, мінімальну кількість об'єктів у листовому вузлі, максимальну кількість листків та інші критерії, які регулюють розмір дерева.

2. Post-Pruning (пост-обрізання) – включає обрізання дерева після того, як воно вже побудоване. Після побудови дерева алгоритм може використовувати валідаційний набір даних або крос-валідацію, щоб визначити, які гілки можна видалити для покращення загальної продуктивності на нових даних [30].

Процедура пост-обрізання може включати в себе такі етапи:

1. Початкова оцінка помилки – вимірюється продуктивність на валідаційному наборі даних до обрізання.

2. Поступове обрізання – починається з низького рівня і поступово видаляються гілки, при цьомунеобхідно спостерігати за зміною продуктивності на валідаційному наборі.

3. Вибір оптимального рівня обрізання – визначення рівня обрізання, який дає найкращий компроміс між «недонавчанням» і «перенавчанням» [31].

Фрагмент згенерованого дерева зображено на рисунку 3.7.

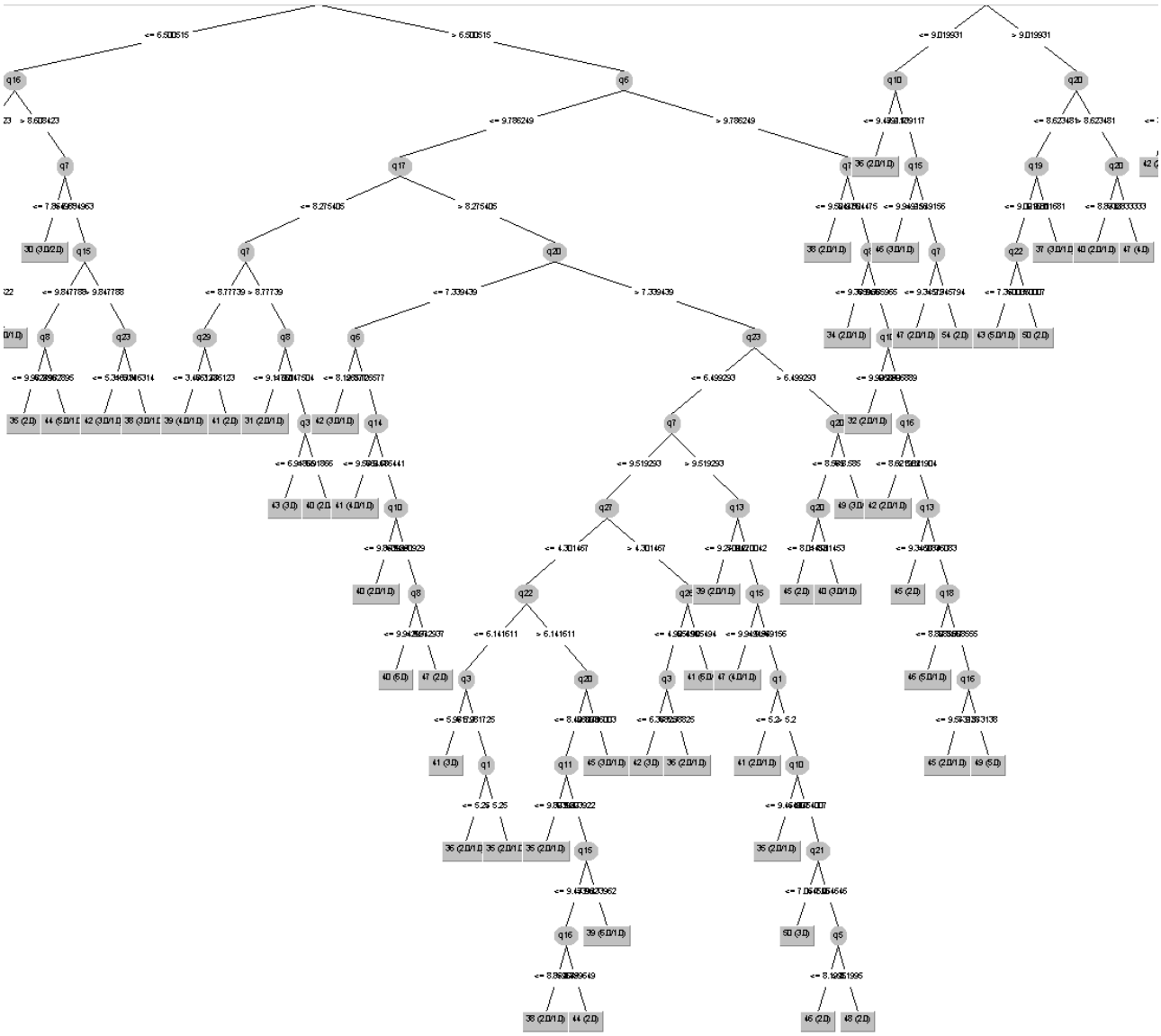


Рисунок 3.7 – Фрагмент згенерованого дерева рішень

Загалом дерево рішень являє собою велику кількість умовних операторів if-then-else, які виконуються на вхідних даних та в результаті ведуть до різних листків-результатів. Завдання моделі – визначити які з вхідних параметрів важливіші, які менш важливі. Таким чином процес класифікації виконується як велика кількість логічних операцій, що забезпечує його високу швидкодію, оскільки логічні операції порівняння є «природніми» для процесорів.

Алгоритм J48 передбачає наступні основні етапи навчання моделі:

1. Вибір атрибута – на кожному кроці алгоритм вибирає атрибут, який найкраще розділяє набір даних на основі якості розділення (зазвичай використовується ентропія або індекс Джині).

2. Розділення даних – дані розділяються на дві або більше групи відповідно до значень вибраного атрибута.

3. Рекурсивне викликання – процес повторюється для кожної нової групи даних, що веде до створення вузлів дерева.

4. Листя – коли досягається критерій зупинки (наприклад, всі об'єкти у вузлі мають однаковий клас, або вже досягнута максимальна глибина дерева), створюється листок, який визначає класифікаційний результат.

Текстове представлення навченої моделі зображено на рисунку 3.8.

```
q21 <= 4.40402
|   q21 <= 1.491978
|   |   q28 <= 1.444784
|   |   |   q5 <= 4.444982
|   |   |   |   q29 <= 1.207066
|   |   |   |   |   q1 <= 5
|   |   |   |   |   |   q9 <= 3.188807
|   |   |   |   |   |   |   q29 <= 1.000661
|   |   |   |   |   |   |   |   q30 <= 1.004741
|   |   |   |   |   |   |   |   |   q3 <= 3.912804
|   |   |   |   |   |   |   |   |   |   q5 <= 3.498543
|   |   |   |   |   |   |   |   |   |   |   q23 <= 1.029528
|   |   |   |   |   |   |   |   |   |   |   |   q22 <= 1.039755
|   |   |   |   |   |   |   |   |   |   |   |   |   q19 <= 1.050703
|   |   |   |   |   |   |   |   |   |   |   |   |   |   q19 <= 1.009709
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   q24 <= 1.02
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   q1 <= 4.310345
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   q1 <= 4.201681
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   q10 <= 1.814701: 1 (4.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   q10 > 1.814701: 0 (2.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   q1 > 4.201681: 1 (8.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   q1 > 4.310345
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   q27 <= 1.018349
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   q17 <= 1.0761
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   q5 <= 2.864294: 0 (2.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   q5 > 2.864294: 1 (6.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   q17 > 1.0761: 0 (5.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   q27 > 1.018349: 0 (2.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   q24 > 1.02: 0 (2.0/1.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   q19 > 1.009709: 1 (4.0/1.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   |   q19 > 1.050703: 0 (2.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   q22 > 1.039755: 0 (2.0)
|   |   |   |   |   |   |   |   |   |   |   |   q23 > 1.029528: 1 (2.0)
|   |   |   |   |   |   |   |   |   |   |   q5 > 3.498543: 1 (13.0)
|   |   |   |   |   |   |   |   |   |   q3 > 3.912804
|   |   |   |   |   |   |   |   |   |   |   q10 <= 2.756839
|   |   |   |   |   |   |   |   |   |   |   |   q1 <= 4.347826
|   |   |   |   |   |   |   |   |   |   |   |   |   q3 <= 4.096735: 0 (3.0/2.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   q3 > 4.096735
|   |   |   |   |   |   |   |   |   |   |   |   |   |   q8 <= 3.1526
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   q4 <= 3.850798
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   q18 <= 1.107169: 3 (4.0/1.0)
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   q18 > 1.107169: 1 (2.0)
```

Рисунок 3.8 – Текстове представлення навченої моделі алгоритму J48

Дане представлення чітко вказує, що найвагомішим параметром для прийняття рішення, модель виділила відповідь на завдання №21, оскільки дерево

починається саме з нього. Далі йде ієрархія умовних операторів над іншими відповідями на тестові завдання, аж до прийняття кінцевого рішення.

Weka дозволяє візуалізувати помилки класифікації, що допомагає оцінити успішність навчання моделі та правильність її роботи. Така візуалізація для навченої вище моделі подана на рисунку 3.9.

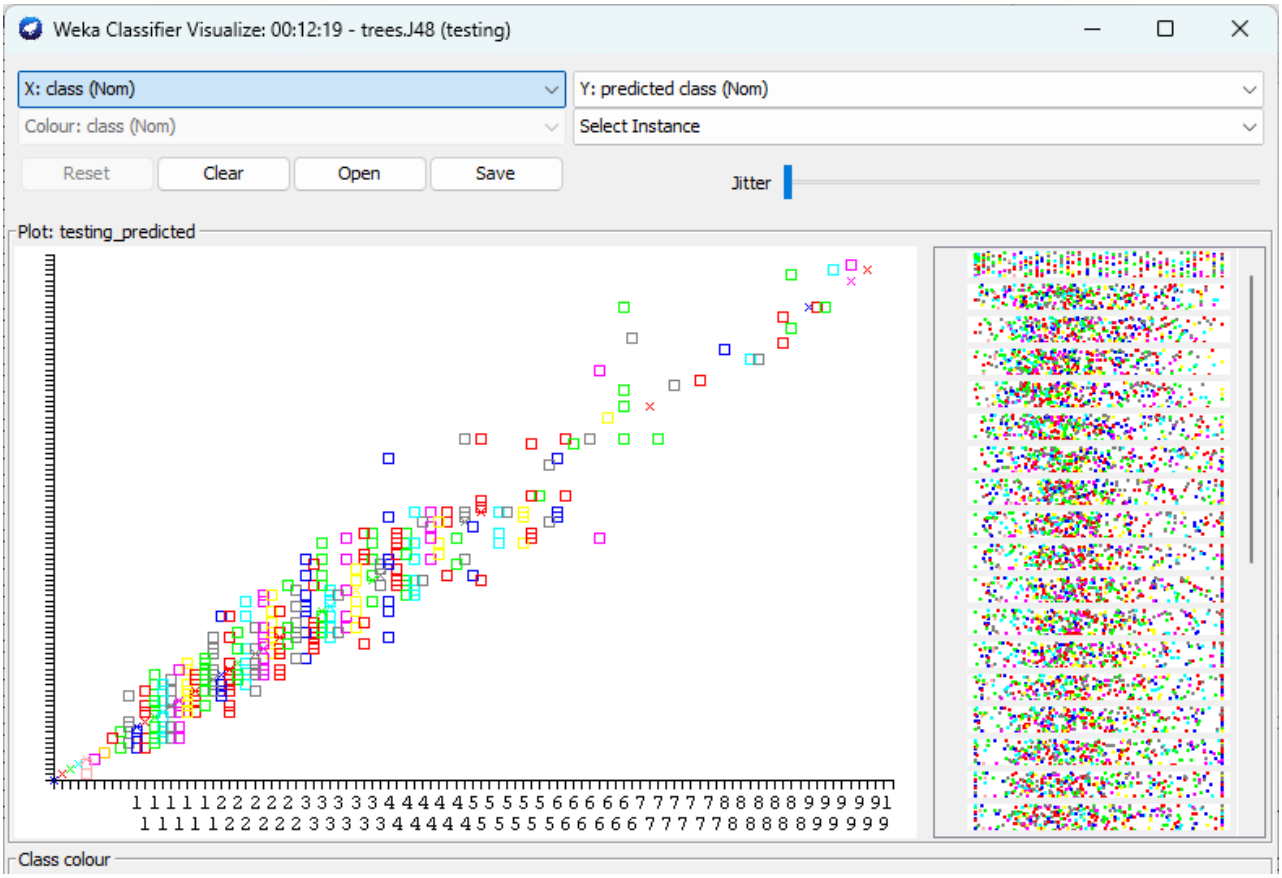


Рисунок 3.9 – Візуалізація помилок класифікації для моделі дерева рішень (J48)

Кожен алгоритм класифікації має різноманітні параметри навчання. Підлаштовуючи ці параметри можна досягнути найкращих результатів навчання. Основні параметри для алгоритму дерева рішень J48:

- binarySplits (true/false) – вказує, чи використовувати двійкові або множинні розбиття для категоріальних атрибутів;

- confidenceFactor (значення від 0 до 1) – вказує мінімальне значення впевненості для обрізання (pruning) дерева: якщо впевненість менше, ніж вказане значення, то гілка обрізається;

- minNumObj (ціле число) – мінімальна кількість об'єктів, яка потрібна на

кожному листку дерева;

- numFolds (ціле число) – кількість складових для перехресної перевірки під час побудови дерева;

- reducedErrorPruning (true/false) – вказує, чи використовувати метод обрізання зменшення помилок;

- saveInstanceData (true/false) – вказує, чи тримати дані прикладів у кожному вузлі;

- subtreeRaising (true/false) – вказує, чи піднімати гілку дерева, якщо це може покращити його взаємодію з іншими гілками;

- unpruned (true/false) – вказує, чи використовувати непідвідений варіант дерева.

Налаштування цих параметрів зображено на рисунку 3.10.

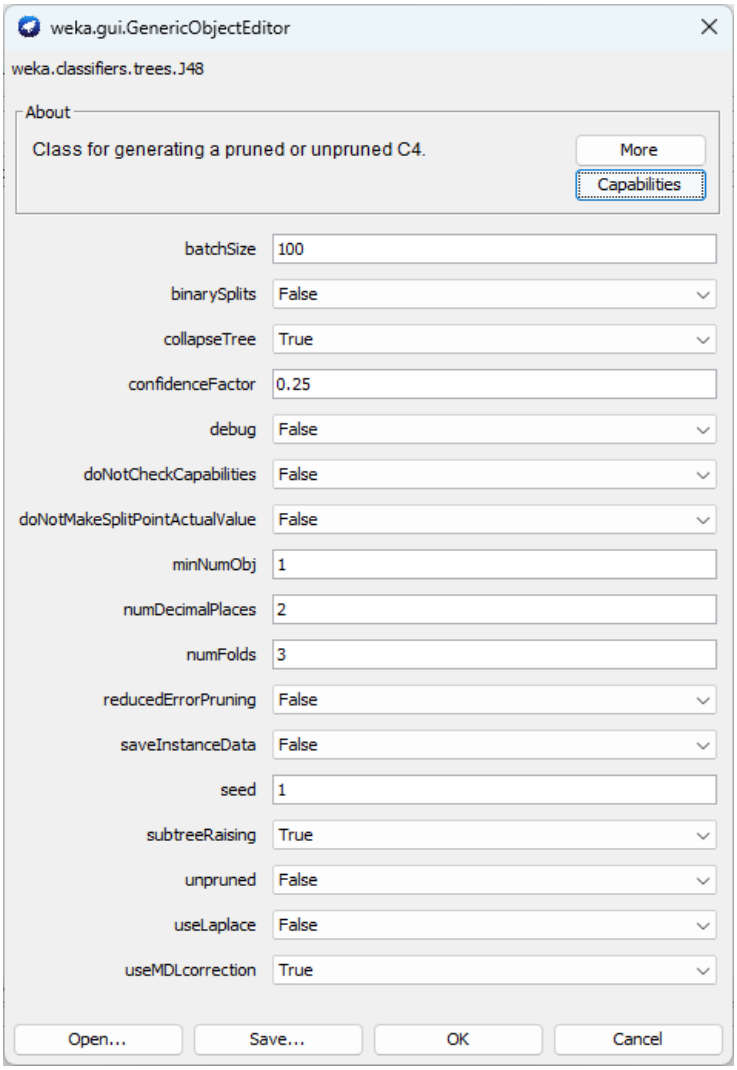


Рисунок 3.10 – Параметри навчання алгоритму дерева рішень J48

Статистична інформація після тестування моделі, яка зображена у Weka, дозволяє оцінити якість навчання і доцільність використання моделі. Модель класифікаційного дерева, навчена вище має всього 11.4% успішних класифікацій із 1000 тестових наборів, що свідчить про низьку якість навчання. Рисунок статистичної інформації наведено на рисунку 3.11.

```
Time taken to test model on test split: 0.05 seconds

=== Summary ===

Correctly Classified Instances         114               11.4     %
Incorrectly Classified Instances       886               88.6     %
Kappa statistic                          0.0976
Mean absolute error                      0.0175
Root mean squared error                  0.1324
Relative absolute error                 90.1985 %
Root relative squared error            134.269  %
Total Number of Instances             1000

=== Detailed Accuracy By Class ===
```

Рисунок 3.11 – Статистична інформація моделі, навченої за алгоритмом J48

Таку низьку якість класифікації можна пояснити неякісною навчальною вибіркою. Проаналізувавши навчальні дані (рисунок 3.12) можна побачити, що вони не збалансовані, тобто найбільше навчальних випадків із результатами тестування від 10 до 40% - таких сценаріїв 90%, тому модель не «розуміє» як працювати з даними, які повинні класифікуватись, наприклад, у оцінку 70-90%.

Однією з вирішальних вимог до машинного навчання, особливо для задач класифікації є перевірка розподілу та балансу класів. Ця вимога означає, що класи у навчальних вибірках повинні бути розподілені рівномірно за кількістю вибірок. Як альтернативу використовують також методи балансу класів такі, як:

- вибір справедливого навчального набору – коли випадковим чином видаляється певна кількість зразків для найбільш представленого класу, або додається певна кількість копій для зразків найменш представленого класу;
- використання ваг класів – присвоєння коефіцієнту ваги кожному класу в залежності від кількості зразків, які його представляють;
- використання метрик якості – використання метрик таких, як F1-мера,

або AUC-ROC, які враховують якість класифікації для обох класів, незалежно від кількості зразків що їх представляють;

- генеративні методи – створюють штучні зразки для класів, які менш-представлені на основі існуючих зразків;

- крос-валідація – забезпечую те, щоб в кожному згині (fold) крос-валідації розподіл класів був подібним до розподілу класів в повному наборі даних [32].

Розподіл навчальної вибірки по класах наведено на рисунку 3.12.

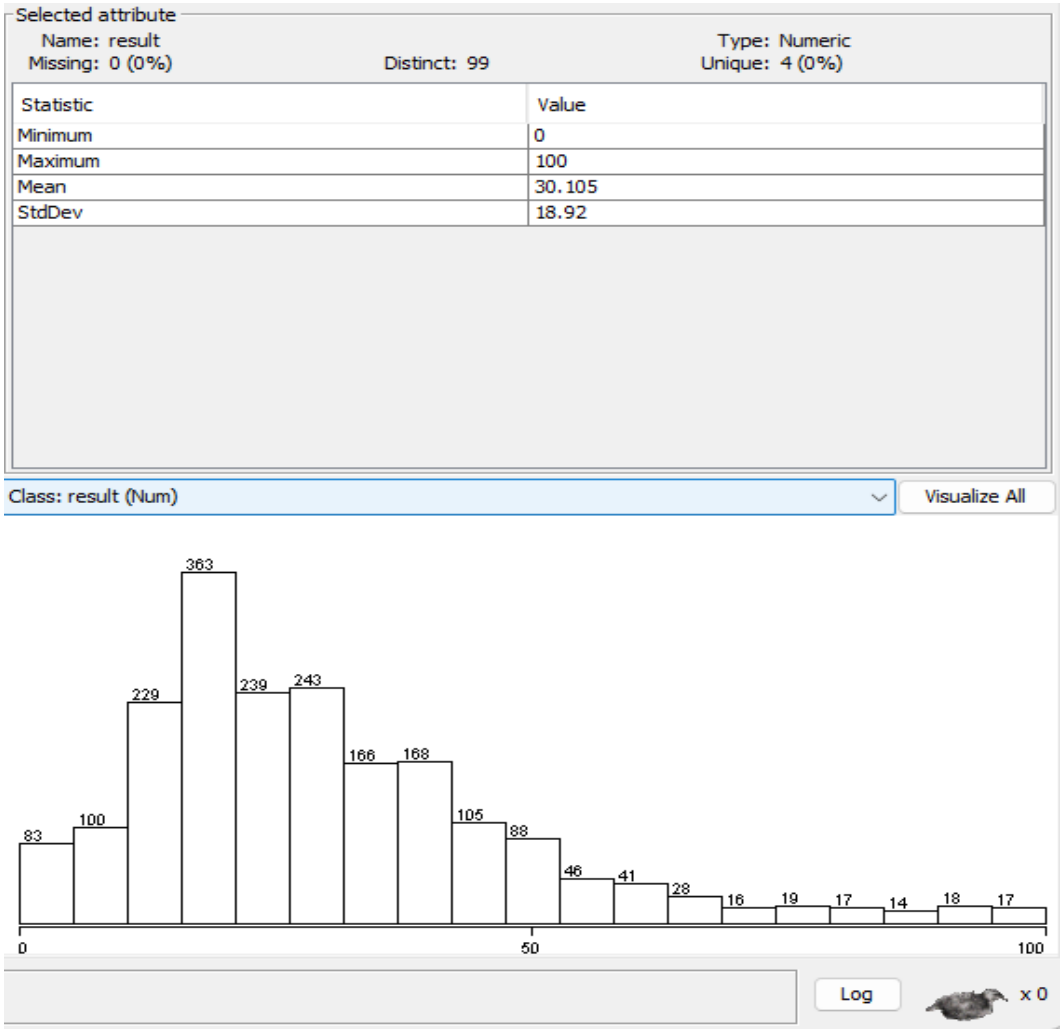


Рисунок 3.12 – Початковий розподіл вхідних даних по очікуваному класу

Процес підготовки даних є ітеративним процесом, і може змінюватися та повторюватись в залежності від характеру даних та конкретного завдання, допоки не буде досягнуто ідеального компромісу в кількості навчальних даних

та якості класифікації отриманої моделі. Тому аналіз даних, як наука, підходить до підготовки вхідних даних максимально ретельно. Після балансування розподілу набору навчальних даних, можна побачити що графік розподілу класів став більш рівномірним, і приблизно для всіх варіацій класів є однакова кількість вхідних даних. Розподіл вхідних даних після балансування наведено на рисунку 3.13.

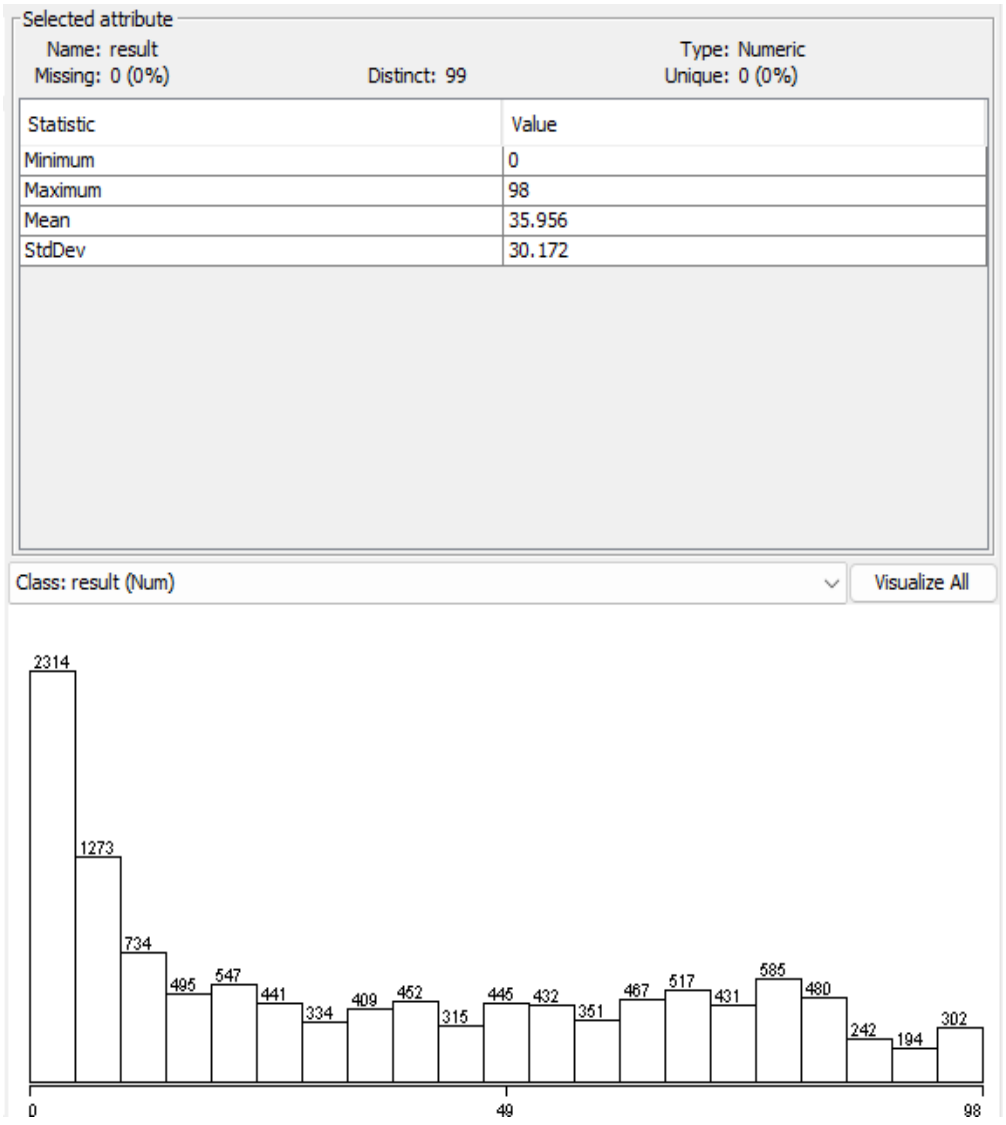


Рисунок 3.13 – Розподіл вхідних даних по очікуваному класу для збалансованих даних

Після перенавчання моделі за алгоритмом класифікаційного дерева отримуємо значно кращий результат класифікації на рівні 25%. Результати тестування наведені на рисунку 3.14.

```
=== Evaluation on test split ===

Time taken to test model on test split: 0.34 seconds

=== Summary ===

Correctly Classified Instances        1761               24.9575 %
Incorrectly Classified Instances      5295               75.0425 %
Kappa statistic                          0.2367
Mean absolute error                      0.0151
Root mean squared error                  0.1128
Relative absolute error                 77.378  %
Root relative squared error            114.3043 %
Total Number of Instances             7056

=== Detailed Accuracy By Class ===
```

Рисунок 3.14 – Статистична інформація моделі, навченої після нормалізації вхідних даних

Розподіл помилок класифікації є прийнятний для поставленої задачі. Максимальне відхилення від очікуваного класу складає 4.5%. Для системи тестування знань похибка в результаті між 66 і 69 є прийнятною. Розподіл помилок наведено на рисунку 3.15.

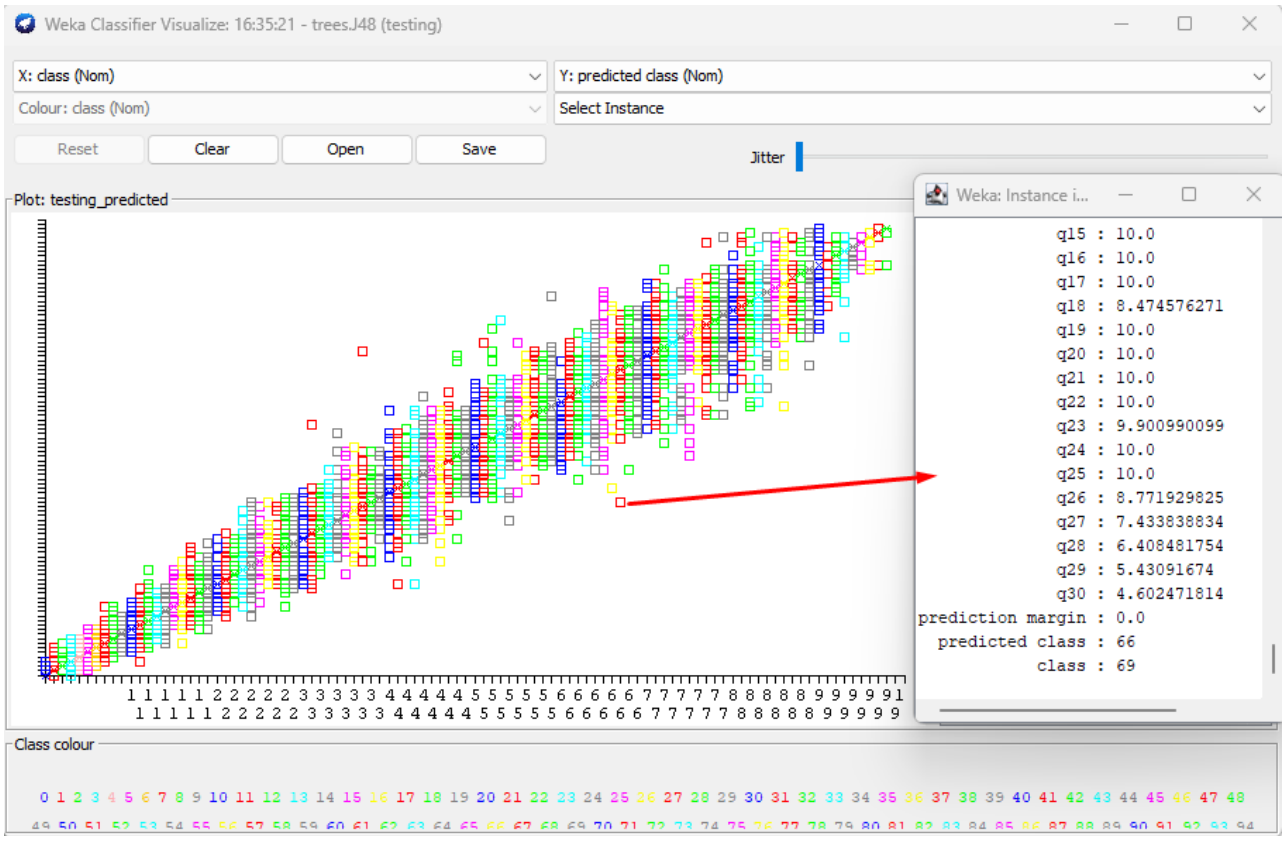


Рисунок 3.15 – Розподіл помилок класифікації для моделі дерева рішень (J48)

Інші типи алгоритмів класифікації показали гірші результати тестування на цьому ж наборі вхідних даних. Так алгоритм пошуку найближчих сусідів показав результат успішної класифікації 21.2%, а наївний Байєсівський класифікатор – 23.2%. Результати згаданих алгоритмів наведено на рисунках 3.16 і 3.17.

```
=== Evaluation on test split ===

Time taken to test model on test split: 2.76 seconds

=== Summary ===

Correctly Classified Instances        1249               21.2415 %
Incorrectly Classified Instances      4631               78.7585 %
Kappa statistic                          0.1988
Mean absolute error                      0.0157
Root mean squared error                  0.1238
Relative absolute error                 80.4409 %
Root relative squared error            125.5066 %
Total Number of Instances             5880
```

Рисунок 3.16 – Статистична інформація моделі за алгоритмом найближчих сусідів

```
=== Evaluation on test split ===

Time taken to test model on test split: 0.29 seconds

=== Summary ===

Correctly Classified Instances         546               23.2143 %
Incorrectly Classified Instances      1806               76.7857 %
Kappa statistic                          0.2188
Mean absolute error                      0.0155
Root mean squared error                  0.1035
Relative absolute error                 79.4598 %
Root relative squared error            104.9475 %
Total Number of Instances             2352
```

Рисунок 3.17 – Статистична інформація моделі за алгоритмом Байєсівського класифікатора

Як видно з результатів тестування, класифікаційне дерево показало найкращий результат в даному випадку, тому саме ця модель буде використана для адаптивної системи тестування знань.

Для моделі класифікаційного дерева важливою умовою успішного навчання є запобігання «перенавчання», тому із підготовленого набору даних в

12 тисяч варіацій, для навчання було задіяно лише 5 тисяч, решта 7 тисяч використовувалась для тестування моделі. Така методика називається обрізка гілок класифікаційного дерева і позитивно впливає на здатність моделі класифікувати невідомі набори даних [33].

Після навчання модель можна експортувати у файл для подальшого використання. Процес збереження моделі зображено на рисунку 3.18

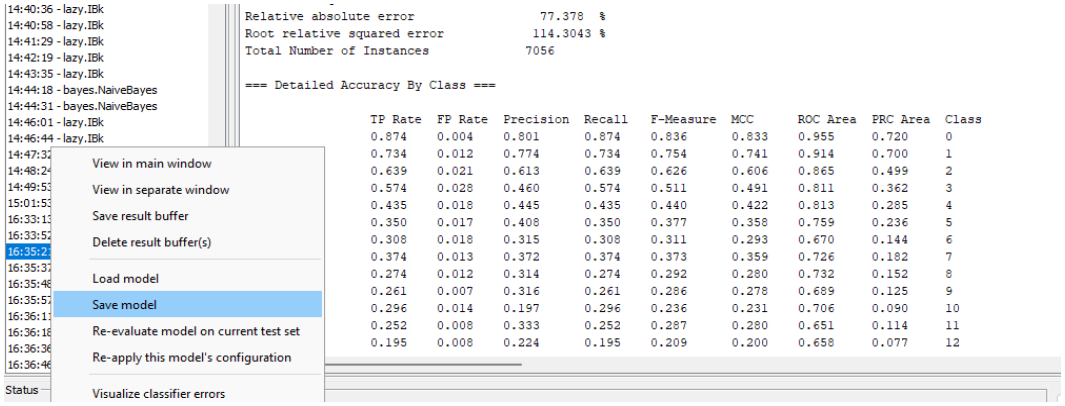


Рисунок 3.18 – Збереження навченої моделі

Збережений файл у форматі .model може бути використаний у різних мовах програмування для проведення класифікації даних.

3.5 Висновки

Навчання моделі штучного інтелекту – це процес, під час якого модель вчиться визначати закономірності у наборі даних і вирішувати завдання без явного програмування. Цей процес включає в себе кілька ключових етапів: збір даних, їх підготовку, вибір алгоритму навчання, безпосереднього навчання, оцінку її результатів, налаштування та перенавчання (якщо потрібно). В результаті проведених операцій в даному розділі було обрано найбільш підходящий алгоритм навчання моделі, підготовлено та нормалізовано вхідні набори даних, проведено навчання та аналіз результатів тестування. Як результат було отримано робочу модель штучного інтелекту для використання в системі адаптивного тестування знань.

4 РОЗРОБКА ПРОГРАМНОГО ЗАСОБУ ДЛЯ АДАПТИВНОГО ТЕСТУВАННЯ ЗНАНЬ

4.1 Обґрунтування способів та засобів реалізації програмного забезпечення для адаптивного тестування знань

Для отримання всіх переваг проектування системи за допомогою інструментального програмного забезпечення було порівняно такі відомі об’єктно-орієнтовані мови програмування: C++, Java, C#.

Усі ці мови дозволяють користуватись перевагами об’єктно-орієнтованого програмування не лише на етапах проектування і конструювання програмних засобів, але і на етапах їхньої реалізації, тестування, супроводу [34].

Нині найбільш розповсюдженою об’єктно-орієнтованою мовою програмування є C++. Вільно розповсюджувані комерційні системи програмування C++ існують практично на будь-якій платформі. Мова Java розроблена компанією Sun Microsystems саме тоді, коли розвиток Інтернету вимагав розосереджених обчислень. Це і стало причиною її масовому впровадженню та популярності. C# є спорідненою мовою програмування до Java. Це теж об'єктно-орієнтована мова програмування з безпечною системою типізації орієнтована на платформи .NET [35, 36]. Результати порівняльного аналізу наведено в таблиці 3.1.

Таблиця 4.1 – Порівняльний аналіз мов програмування

Параметр	C#	C++	Java
Безпечна типізація	+	-	+
Автоматичне вивільнення пам’яті	+	-	+
Перевантаження операторів	+	+	-
Повна сумісність з операційними системами Windows	+	-	-

Оскільки платформа .NET має ідеальну сумісність з операційною системою Windows, має автоматичне управління пам’яттю а мова C# потужно розвивається з кожним роком, було обрано саме ці технології для реалізації

програмного забезпечення для адаптивного тестування знань [37].

У якості середовища розробки було обрано MS Visual Studio, оскільки це середовище містить усі необхідні компоненти для розробки мовою C# та має інтегровану сумісність з Visual Paradigm [38].

Для зв'язку серверної та клієнтської частини програмного забезпечення було обрано WCF (Windows Communication Foundation) – технологію, розроблену Microsoft для створення розподілених інтерфейсів служб (services) в середовищі Windows. WCF дозволяє розробляти служби, які можуть взаємодіяти між собою за допомогою різних протоколів, таких як HTTP, TCP, інші протоколи моделі OSI, а також підтримує різні формати обміну даними, такі як XML та JSON. Основні характеристики та можливості технології WCF включають:

- протоколи та формати обміну даними: WCF підтримує різні протоколи для обміну даними, такі як HTTP, TCP, MSMQ, і інші. Дані можуть бути представлені в різних форматах, таких як XML, JSON і бінарні формати;

- безпека: WCF надає вбудовану підтримку для різних механізмів безпеки, включаючи ідентифікацію та автентифікацію, авторизацію, шифрування та підпис;

- інтеграція з іншими технологіями Microsoft: WCF легко інтегрується з іншими технологіями Microsoft, такими як ASP.NET, Windows Workflow Foundation (WF), та Windows Identity Foundation (WIF);

- широкий спектр розподілених служб: WCF може бути використаний для створення різних видів розподілених служб, включаючи служби для обміну повідомленнями, служби для передачі потоку даних, служби для створення та керування транзакціями, та інші;

- підтримка асинхронного програмування: WCF дозволяє використовувати асинхронний код для реалізації ефективної обробки багатозадачних операцій та подій;

- метадані та інструменти для розробників: WCF надає можливість створення метаданих для служб, що дозволяє автоматично генерувати клієнтські частини коду, також інтегрується з інструментами розробки, такими як Visual

Studio, для полегшення розробки і тестування;

- підтримка різних платформ: WCF може бути використаний для створення служб, які взаємодіють з різними платформами, включаючи ті, що використовують технології, які не є Microsoft-centric;

- розширюваність: WCF є розширюваною технологією, що дозволяє розробникам використовувати власні розширення та налаштування для задоволення конкретних потреб.

Технологія WCF надає розробникам гнучкість і потужність для створення розподілених додатків та служб у середовищі Microsoft [39].

Для зберігання реляційних даних використано СУБД Microsoft SQL Server. Це популярна система, має ряд переваг, завдяки чому вона є вибором багатьох організацій. Основні переваги MS SQL Server:

- надає відмінні опції масштабування, дозволяючи користувачам розширювати або масштабувати свої бази даних. Він підтримує як вертикальне масштабування (додавання більше ресурсів до одного сервера), так і горизонтальне масштабування (розподілену архітектуру баз даних);

- включає різноманітні функції оптимізації продуктивності, такі як індексація, оптимізація запитів та обробка в пам'яті. Також підтримується паралельна обробка, яка може значно покращити продуктивність запитів;

- має надійні засоби безпеки для захисту даних. Підтримується шифрування, механізми аутентифікації та керування доступом;

- «природньо» інтегрується з іншими продуктами та сервісами Microsoft, такими як хмарні служби Azure, Power BI для бізнес-аналітики та Visual Studio для розробки. Ця інтеграція полегшує впровадження SQL Server у інфраструктури організацій, які вже використовують технології Microsoft;

- висока доступність та відновлення після аварії: SQL Server надає можливості високої доступності, такі як Always On Availability Groups та реплікацію баз даних, для забезпечення стійкості та відновлення після аварій. Ці можливості дозволяють автоматично переключати обслуговування, реплікувати дані та використовувати стратегії резервного копіювання для мінімізації

простою;

- підтримує кілька мов програмування, таких як T-SQL, мови .NET та XML. Надається ряд інструментів розробки, таких як SQL Server Management Studio (SSMS) та Visual Studio, для полегшення розробки програм та управління базами даних [40, 41].

В якості середовища для проектування та створення бази даних було обрано оболонку MS SQL Server Management Studio – оскільки це «рідна» оболонка від Microsoft, яка гарантовано підтримує всі можливості бази даних MS SQL Server [42].

Для створення графічного інтерфейсу було обрано технологію WPF, яка дозволяє створювати динамічні інтерфейси для операційної системи Windows.

4.2 Розробка ER-моделі бази даних

Модель сутності-зв'язок (ER-модель) є методом візуалізації та опису даних у базах даних. Вона використовує концепції сутностей, атрибутів і зв'язків для представлення структури даних. Основні елементи ER-моделі включають:

- сутності (Entities) – представляють об'єкти або поняття, які мають важливе значення і можуть бути ідентифіковані. Наприклад, "Клієнт", "Продукт" або "Замовлення" можуть бути сутностями;

- атрибути (Attributes) – визначають властивості або характеристики сутності: наприклад, у сутності "Клієнт" можуть бути атрибути, такі як ім'я, прізвище, адреса;

- зв'язки (Relationships) – вказують на зв'язок між сутностями: наприклад, зв'язок між "Клієнтом" і "Замовленням" може вказувати, що клієнт може робити багато замовлень, а кожне замовлення належить одному клієнту;

- ключі (Keys) – визначають унікальний ідентифікатор для сутностей і допомагають встановити унікальність записів у таблицях бази даних;

- слабкі сутності (Weak Entities) – це сутності, які не можуть бути ідентифіковані без залежності від інших сутностей та зазвичай потребують існування зв'язку з "власником";

- типи зв'язків (Relationship Types) – вказують на характер зв'язку між сутностями, такі, як один до одного, один до багатьох, багато до багатьох;

- кардинальність (Cardinality) – вказує на кількість об'єктів, що беруть участь у зв'язку. Наприклад, 1:1, 1:n, n:n.

ER-модель допомагає аналізувати потреби системи та проектувати базу даних з огляду на її структуру та взаємозв'язки між об'єктами даних. Зазвичай ER-модель використовується як основа для подальшого проектування реляційної бази даних, але вона також може бути використана для представлення інших видів баз даних [42].

В базі даних необхідно забезпечити збереження інформації про про групи, зареєстрованих користувачів, та їх результати проходження тестування. Також потрібно зберігати теми тестування, тести, тестові завдання та варіанти відповідей.

При створенні бази даних, виділено такі сутності: ГРУПА, КОРИСТУВАЧ, РЕЗУЛЬТАТ, ТЕСТОВАНИЙ, ТЕМА, ТЕСТ, ЗАВДАННЯ, ВІДПОВІДЬ, РЕСУРС.

Сутність ГРУПА містить інформацію про зареєстровані групи з метою групування користувачів тестової системи.

Сутність КОРИСТУВАЧ містить інформацію про користувача системи, зокрема ім’я, логін, пароль в системі, ідентифікатор групи та інформацію про його сесію.

Сутність РЕЗУЛЬТАТ містить інформацію про проходження тесту (дата, результат, ідентифікатор тесту, ідентифікатор користувача).

Сутність ТЕМА містить назви всіх створених тем для групування тестових наборів.

Сутність ТЕСТ містить ідентифікатор теми, назву та опис тесту.

Сутність ЗАВДАННЯ представляє одне тестове завдання. Містить ідентифікатор тесту, тип завдання, його опис, та додаткові матеріали.

Сутність РЕСУРС представляє різноманітні ресурси, файли, які можна додавати до тестових завдань.

Сутність ВІДПОВІДЬ являє собою один варіант відповіді на завдання. Містить інформацію про правильність, опис відповіді та ідентифікатор завдання,

Модель сутність-зв'язок наведена на рисунку 4.1.

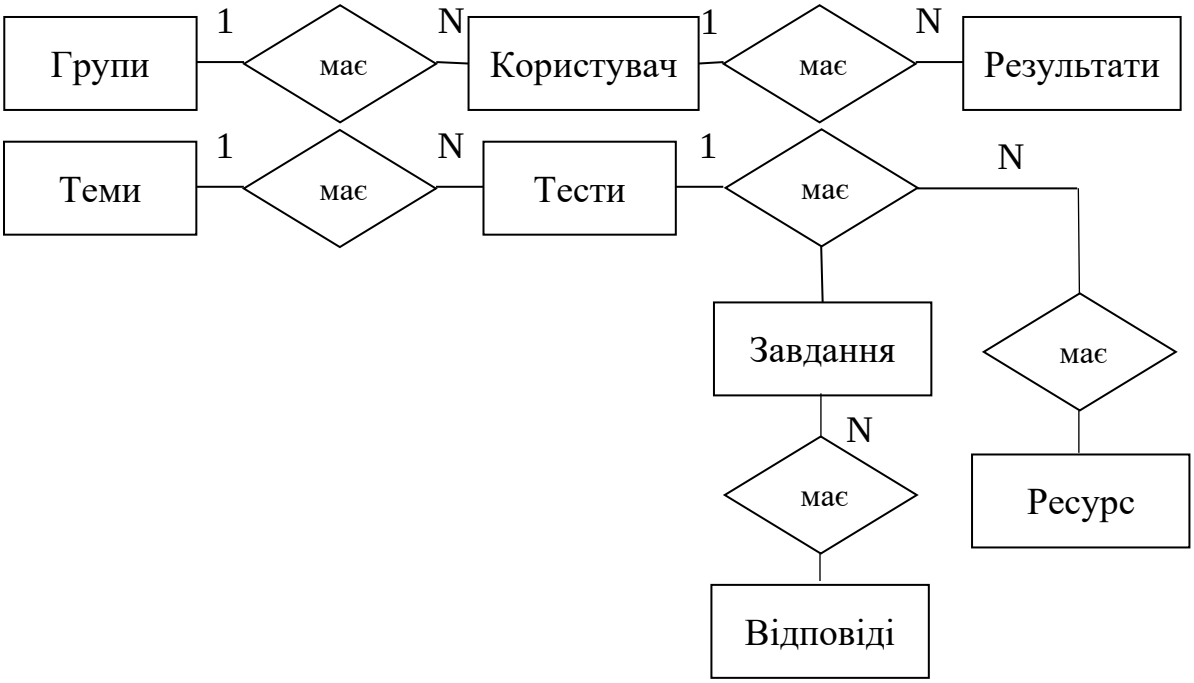


Рисунок 4.1 – ER-модель

База даних спроектована наступним чином:

- ГРУПИ мають СТУДЕНТІВ;
- СТУДЕНТИ мають РЕЗУЛЬТАТИ;
- ТЕМИ мають ТЕСТИ;
- ТЕСТИ мають ЗАВДАННЯ;
- ЗАВДАННЯ мають ВІДПОВІДІ;
- ЗАВДАННЯ мають РЕСУРСИ.

Атрибути сутностей:

- Групи (Ідентифікатор, Назва групи);
- Користувачі (Ідентифікатор, Ім’я, Логін, Пароль, Ідентифікатор групи, Стан);
- Результати(Дата початку, Дата завершення, Результат, Результат у відсотках, Ідентифікатор тесту, Ідентифікатор студента);

- Теми (Ідентифікатор, Назва, Опис);
- Тести (Ідентифікатор, Ідентифікатор теми, Назва, Опис);
- Завдання (Ідентифікатор, Ідентифікатор тесту, Опис, Тип, Складність, Додаткові дані);
- Відповіді (Ідентифікатор, Ідентифікатор завдання, Опис, Додаткові дані, Правильність);
- Ресурси (Ідентифікатор, Контент, Тип).

4.3 Розробка серверної частини

Для оптимізації процесу розробки було використано автоматичну генерацію коду на основі розроблених раніше діаграм за допомогою CASE – засобу Visual Paradigm.

Для генерації вихідного коду необхідно використовувати створені на етапі проектування діаграми класів, які формують «кістяк» програмного засобу, а також діаграми компонентів, які слугують контейнерами для цих класів.

Для мови програмування C# зокрема можна налаштовувати простори імен, автоматичні поля, префікси для атрибутів, вбудовані типи даних для асоціативних зв’язків на діаграмах та інше [43].

Після запуску процесу генерації Visual Paradigm створено C# файли програмного коду, на основі яких було реалізовано програмну частину серверної частини системи адаптивного тестування знань. Реалізацію системи виконано за за допомогою середовища Visual Studio та мови програмування C#.

Відповідно до спроектованої архітектури, серверна частина системи складається з наступних компонентів:

- Application;
- Domain;
- Infrastructure.

Для генерації коду було використано діаграми класів (додаток В), так як вони найбільш повно описують низькорівневі деталі об’єктів системи, їх

функціонал, типи даних. Вікно інструменту Instant Generator, який використано для генерації коду зображено на рисунку 4.2.

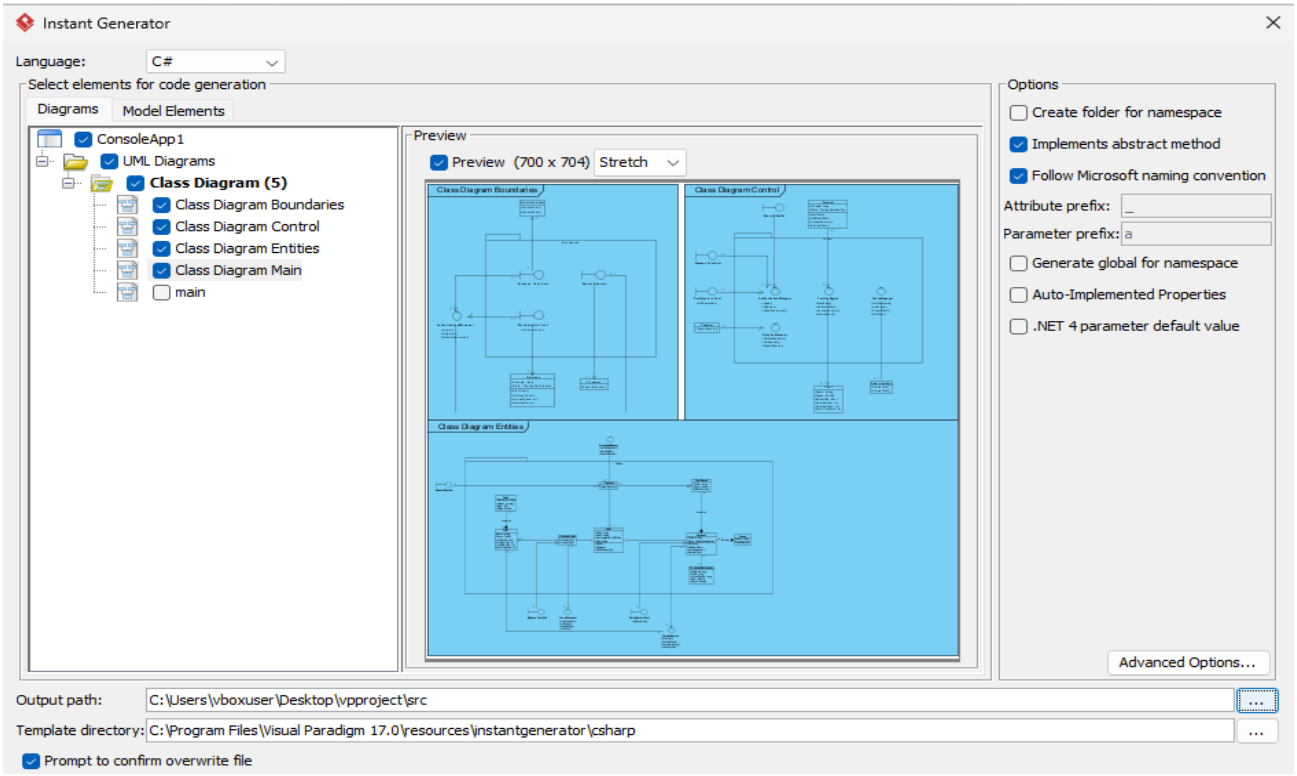


Рисунок 4.2 – Вікно інструменту Instant Generator

Вся керуюча логіка міститься в бібліотеці класів Application.dll. Основні класи, які інкапсулюють логіку системи це:

- AuthorizationManager – клас, що відповідає за реєстрацію та авторизацію користувачів системи;
- ResultsManager – клас, який відповідальний за отримання результатів тестування;
- TestingEngine – клас, який виконує безпосередньо процес тестування. Координує вибір завдання, опрацьовує відповіді користувачів;
- TestsManager – клас, призначений для створення та редагування тестових наборів даних;
- UsersManager – клас, відповідальний за редагування груп користувачів.
- SettingsManager – відповідає за редагування та збереження різноманітних параметрів тестування.

Програмний код основних класів серверної частини наведено в додатку Г. Серверний додаток може використовуватись користувачами різних ролей: адміністратор та екзаменатор. Адміністратор відповідальний за налаштування процесу тестування, відстеження системних повідомлень, укладання нових тестових наборів, створення груп користувачів та ін. Екзаменатор може перевіряти та експортувати результати тестування для конкретних груп користувачів.

Інтерфейс додатку імплементовано на основі технології Windows Forms. Це бібліотека для створення графічних інтерфейсів користувача в програмах, розроблених на мові програмування C# або інших мовах, що підтримують платформу .NET Framework. WinForms є однією з технологій, що входять до складу .NET Framework і надає розробникам можливість створювати вікна, кнопки, текстові поля, таблиці та інші елементи інтерфейсу.

Набір функціоналу, який відображається та доступний користувачу залежить від його ролі. Роль визначається на етапі авторизації. Інтерфейс вікна авторизації наведено на рисунку 4.3.

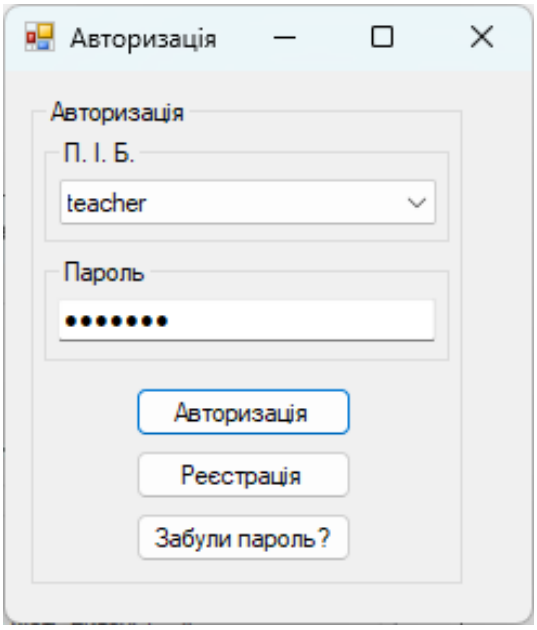


Рисунок 4.3 – Інтерфейс вікна авторизації

Стандартне вікно авторизації дозволяє також зареєструвати нового користувача та відновити пароль при його втраті. Роль користувачеві може

змінити тільки адміністратор після завершення реєстрації. Всі нові реєстрації з серверного додатку за замовчуванням мають роль «Екзаменатор» (Вчитель). Інтерфейс серверного додатку для ролі «Екзаменатор» наведено на рисунку 4.4.

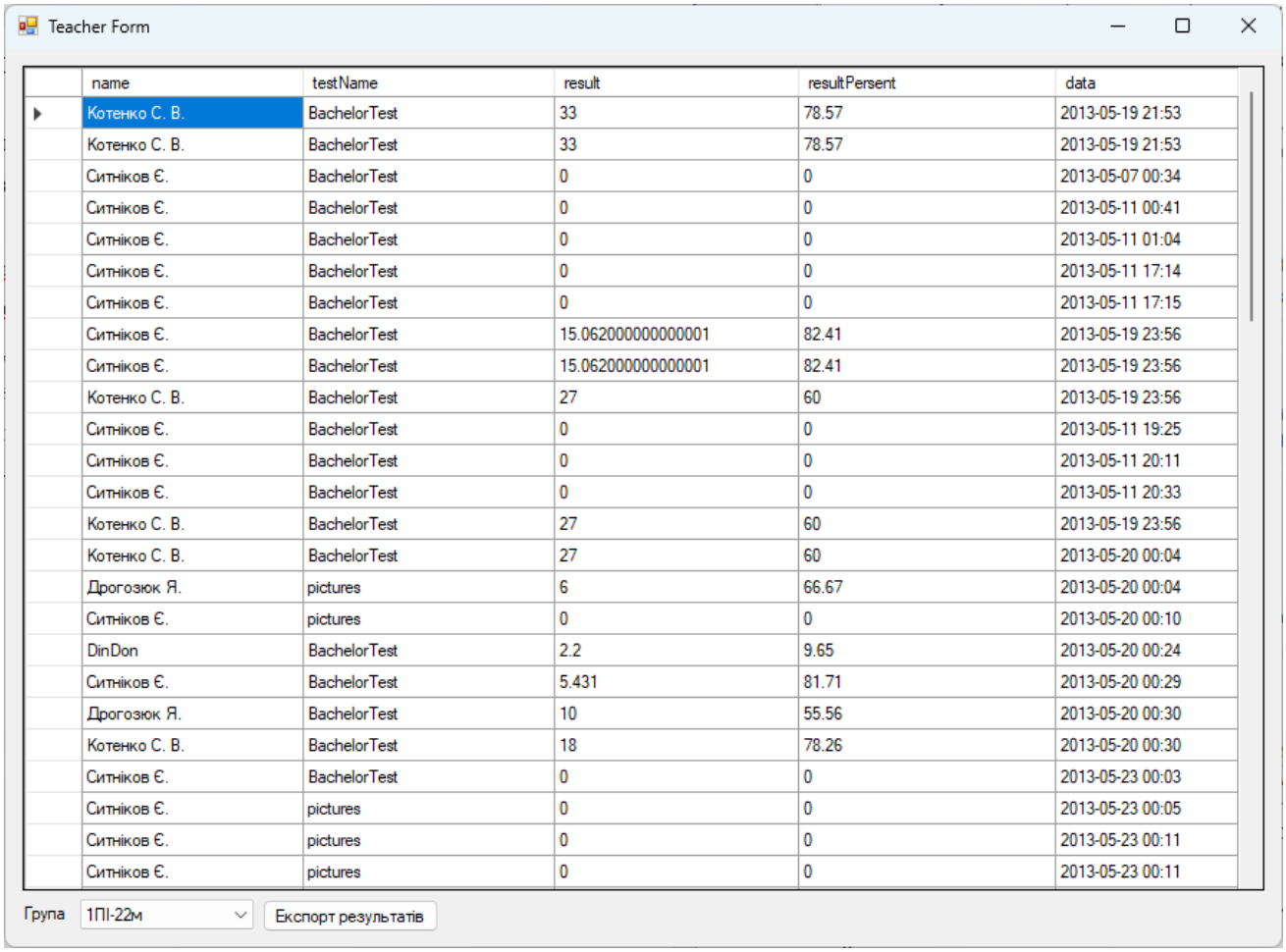


Рисунок 4.4 – Інтерфейс серверного додатку ролі «Екзаменатор»

Користувач даної ролі може переглядати результати тестування окремих груп тестованих, сортувати їх за назвою тесту та експортувати результати у окремий файл csv-формату. CSV (Comma-Separated Values) - це формат текстового файлу, який використовується для представлення табличних даних у вигляді тексту, де кожне значення поля відокремлюється від іншого комою (звідси і назва). Кожен рядок в файлі представляє собою один рядок таблиці, а значення полів можуть бути розділені комами або іншими символами, такими як крапки з комами

Користувач ролі «Адміністратор» бачить зовсім інший інтерфейс

серверного додатку. Його вигляд наведено на рисунку 4.5.

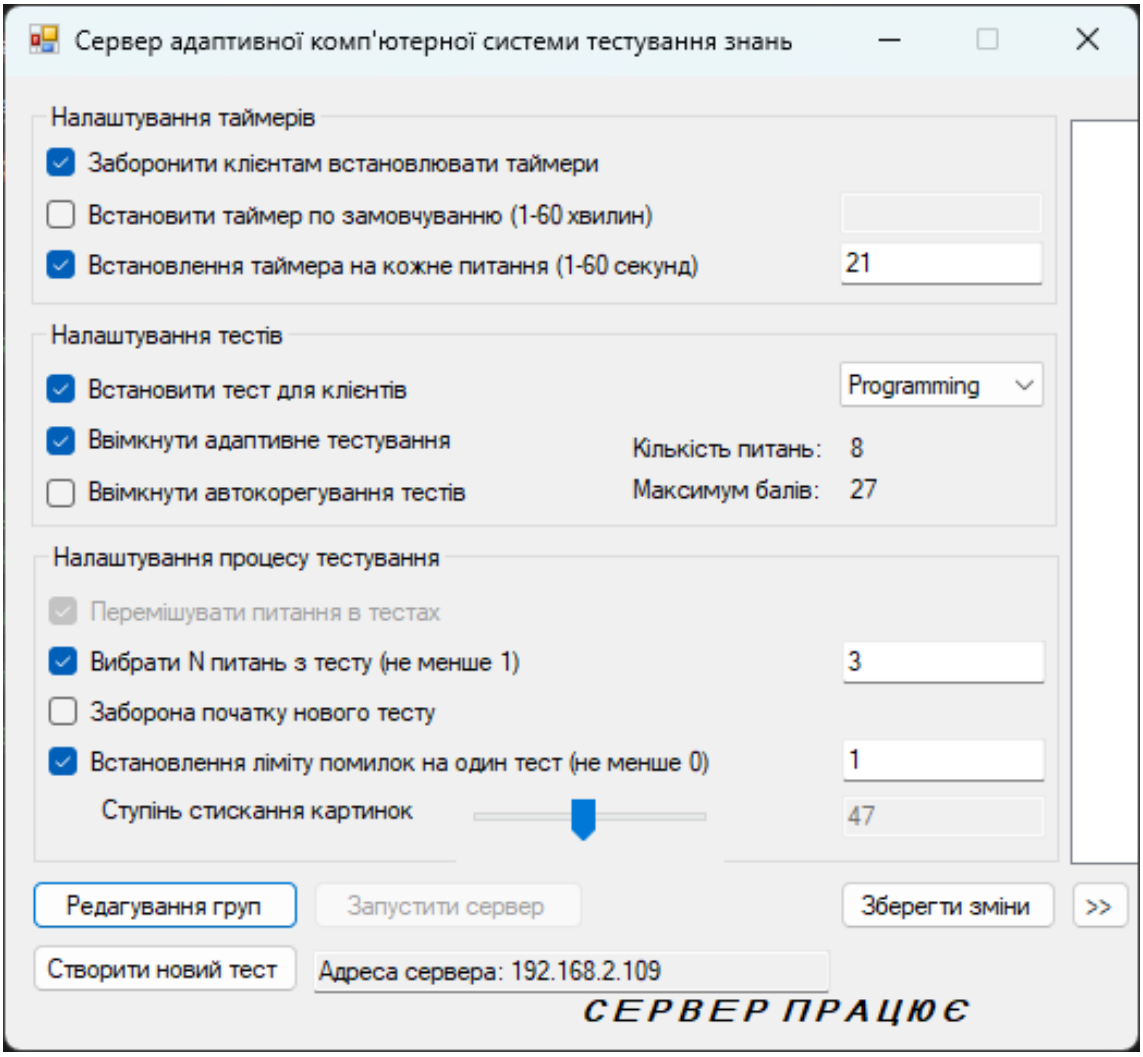


Рисунок 4.5 – Інтерфейс серверного додатку ролі «Адміністратор»

Додаток містить різноманітні налаштування: дозволяє встановлювати таймери, вмикати та вимикати адаптивне тестування, тобто підключати класифікаційну модель, встановлювати ліміт помилок, обирати тест для проходження користувачами та ін.

Крім того інтерфейс дозволяє адміністратору в реальному часі відслідковувати прогрес тестування користувачів у додатковому вікні подій, де логується початок, кінець тестування, авторизація клієнтів та інші події.

Після запуску додатку відображається IP-адреса, яку необхідно законфігурувати у клієнтському додатку для встановлення підключення.

Крім налаштування та управління процесом тестування адміністратор має доступ до створення та редагування тестових завдань. Є можливість обирати стандартні типи завдань: з однією відповіддю, з декількома відповідями, з вводом вільного тексту.

Крім того укладачу тесту необхідно ввести початкову складність завдання. Звичайно це суб'єктивна оцінка, проте в процесі використання системи дана оцінка буде відкалібрована на основі відповідей користувачів. Саме за це відповідає покращений метод «адаптивності» системи. Інтерфейс вікна створення нового тесту наведено на рисунку 4.6.

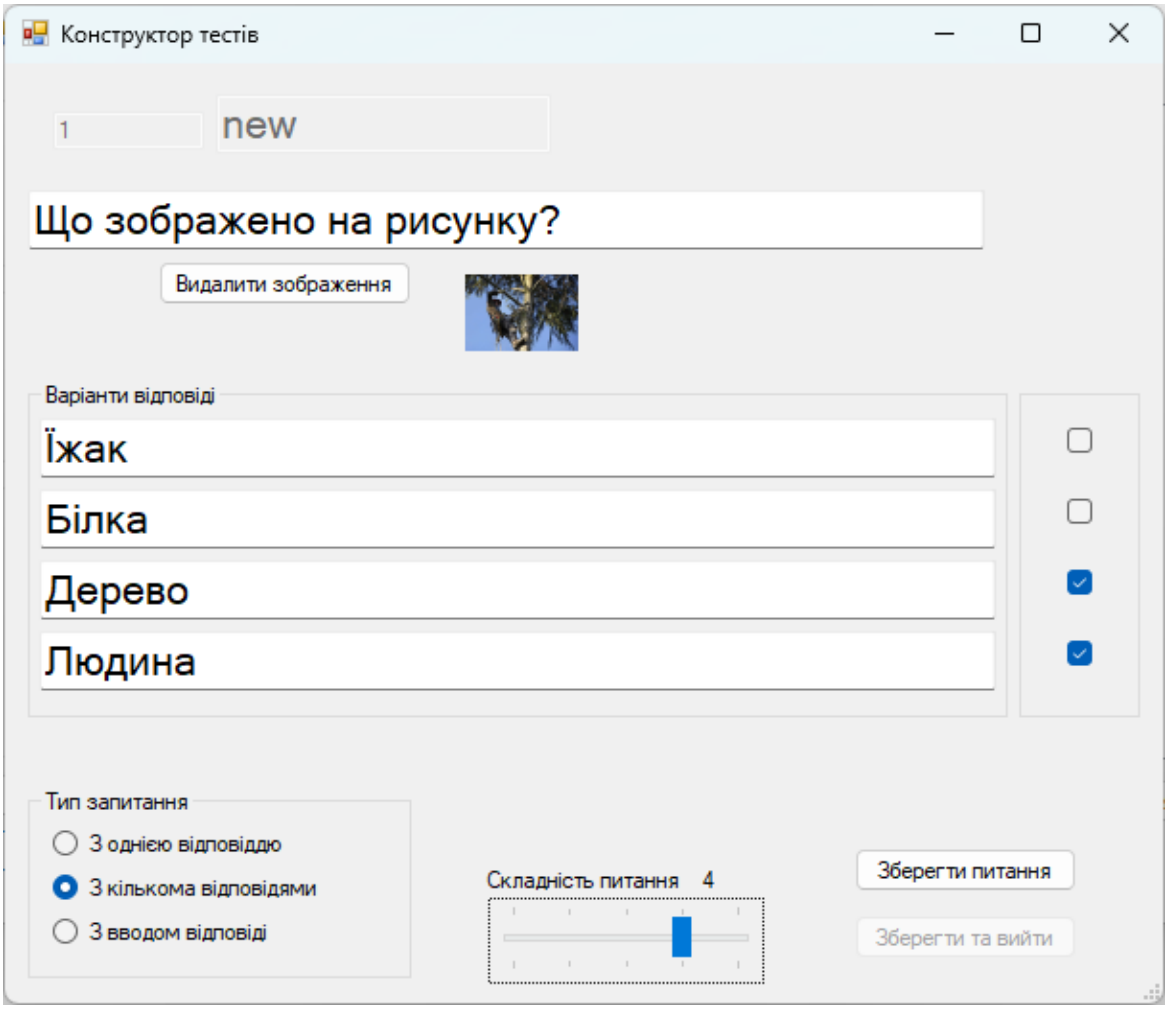


Рисунок 4.6 – Інтерфейс вікна створення нового тесту

Важливим налаштуванням системи є «Ступінь стискання картинок», який забезпечує економію трафіку та може бути корисним при низькій якості мережевого з'єднання між клієнтським та серверним додатками.

4.4 Розробка користувацького інтерфейсу

Оскільки практично вся функціональна частина програми зосереджена на серверній частині, клієнтська програма розроблена з акцентом на користувацький інтерфейс.

Для створення високоякісної апаратно-підтримуваної графіки, використовується технологія WPF. Windows Presentation Foundation – графічна система в складі NET Framework, починаючи з версії 3.0. Дана технологія тісно використовує мову розмітки XAML.

XAML (Extensible Application Markup Language) – це декларативна мова розмітки, що використовується для опису інтерфейсів користувача в технологіях, які базуються на платформі Microsoft, таких як WPF (Windows Presentation Foundation), Silverlight, та інші. Загалом, розмітка XAML визначає розташування об’єктів, панелей, кнопок та інших елементів керування, що складають вікна в додатках WPF.

Для користувацького інтерфейсу було розроблено шаблони основних котролів. Шаблон створеної кнопки наведено на рисунку 4.7.

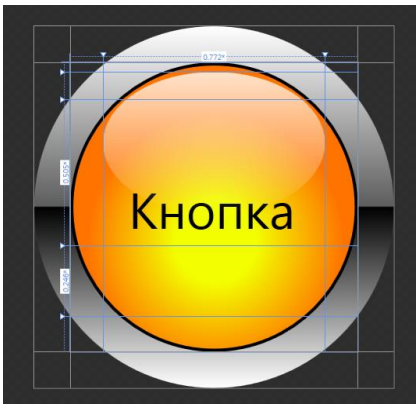


Рисунок 4.7 – Шаблон розробленої кнопки

Мова XAML досить надлишкова, а код отримується досить громіздкий. Вручну писати такий код – це дуже рутинний процес, тому існують різні візуальні оболонки, що автоматично генерують XAML код. В проекті

використано програму Microsoft Expression Blend для розробки користувацького інтерфейсу [44].

Microsoft Expression Blend є інструментом для дизайну і розробки інтерфейсів користувача (UI) для застосунків, створених на платформі Microsoft і використовується головним чином для розробки XAML інтерфейсів для платформи Windows Presentation Foundation (WPF) і Silverlight. Даний інструмент дозволяє додавати візуальні ефекти, анімації і переходи до елементів інтерфейсу користувача без прямого втручання в код. Розроблені в Expression Blend проекти можна легко інтегрувати з середовищем розробки Visual Studio для продовження розробки додатку з використанням розроблених елементів інтерфейсу. Головне вікно програми наведено на рисунку 4.8.

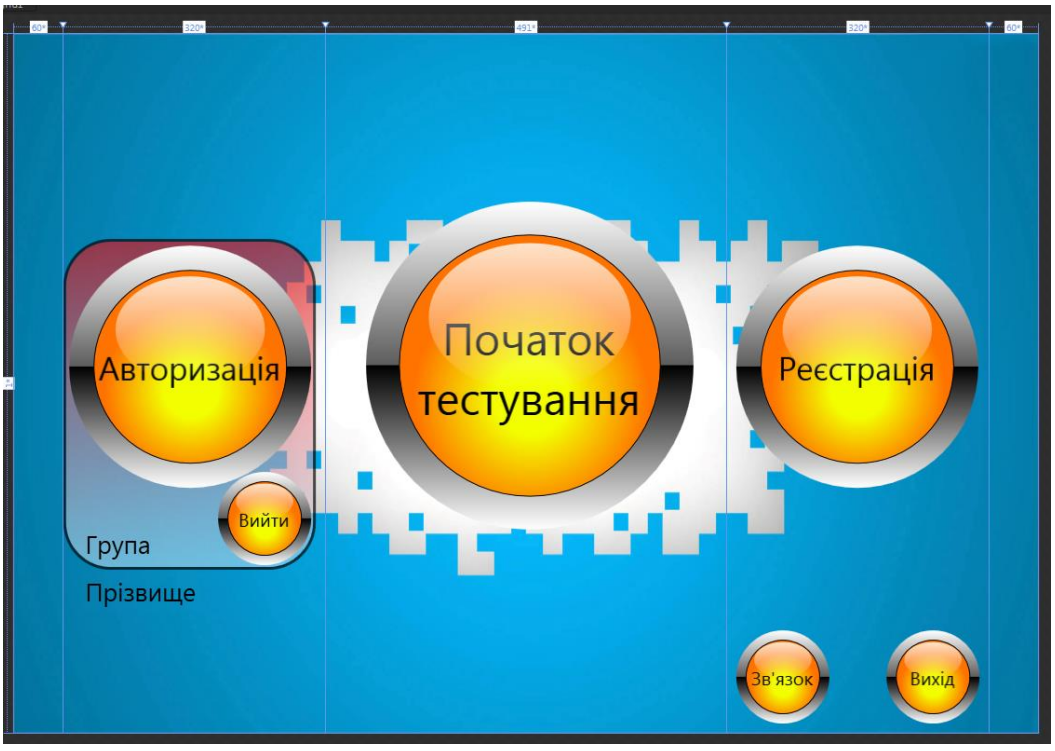


Рисунок 4.8 – Головне вікно програми

В даному проекті не використовуються стандартні вікна з бібліотеки Windows.Forms. Замість того розроблено власні вікна на основі технології WPF.

Особливістю WPF є те, що графічна оболонка описується на мові XAML, а функціональна частина – на іншій, будь-якій мові, що підтримується .NET-платформою. Тому кожне вікно представлено двома наборами коду [45, 46].

Expression Blend дозволяє визначати стилі і шаблони, щоб забезпечити консистентність інтерфейсу в рамках застосунку. Саме ця функція була використана щоб використовувати єдиний шаблон кнопки у всіх вікнах та діалогах додатку.

Завдяки підтримці анімації у WPF, було розроблено механізм виникнення та зникнення діалогових вікон.

Процес використання додатку розпочинається з вікна авторизації чи реєстрації. Інтерфейс діалогу авторизації наведено на рисунку 4.9.

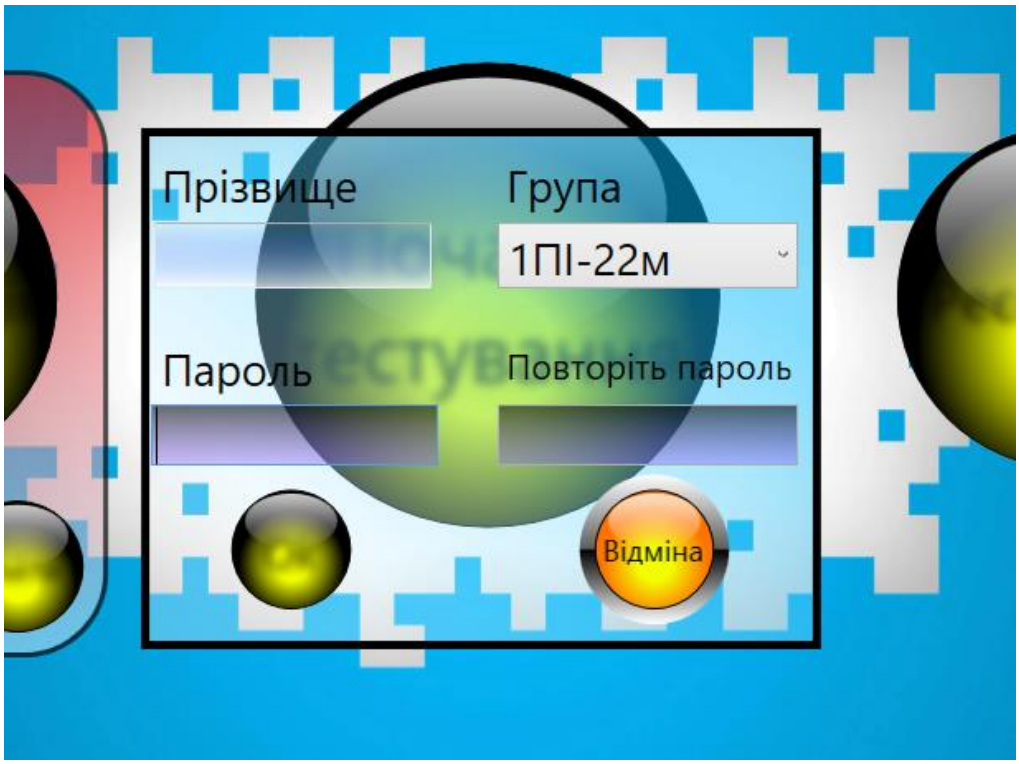


Рисунок 4.9 – Діалогове вікно авторизації

У випадку нового користувача додаток дозволяє відправити запит на серверний додаток та зареєструвати нового користувача з роллю «Тестований».

Крім головного вікна, в проекті також проведено дизайн та розроблено вікно тестування на основі WPF. В залежності від типу завдання відображаються відповідні контроли для вводу/вибору відповіді. Якщо завдання передбачає додаткові ресурси(картинки) – додаток дозволяє розгорнути їх на весь екран для детального перегляду. Інтерфейс вікна тестування подано на рисунку 4.10.

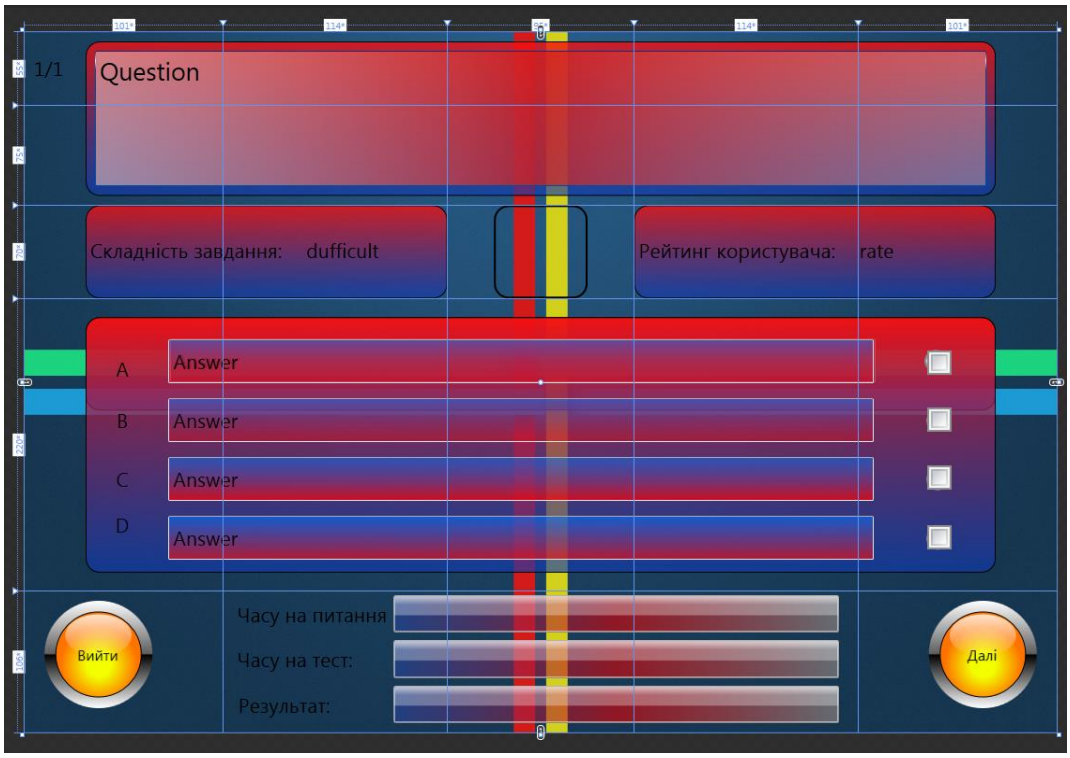


Рисунок 4.10 – Вікно тестування

Використання технології WPF дозволило розробити гнучкий та оригінальний інтерфейс користувача.

4.5 Інтеграція моделі штучного інтелекту

Модель класифікаційного дерева було збережено як файл testingModel.model на попередньому етапі. Даний файл підключено як ресурс до проекту серверного додатку у Visual Studio. При компіляції коду файл моделі копіюється разом зі скомпільованими бінарними файлами.

Такий спосіб підключення дозволяє в майбутньому навчати та підключати нові моделі, покращуючи характеристики системи без необхідності зміни, перекомпіляції коду програми. Оскільки Weka – Java-орієнтована мова, то експортована модель може бути інтегрована та використана лише у java-застосунках.

За допомогою бібліотеки класів NTTU.BigODM.MachineLearning.Weka було інтегровано модель до серверної частини системи тестування знань. Даний

пакет побудований з використанням IKVM.NET що являє собою відкриту реалізація мови програмування Java для платформи .NET. Забезпечує можливість взаємодії між Java і .NET коду, перетворюючи Java bytecode в Common Intermediate Language (CIL) – проміжний мовний код, який використовується платформою .NET. Підключення експортованої моделі до проекту зображено на рисунку 4.11.

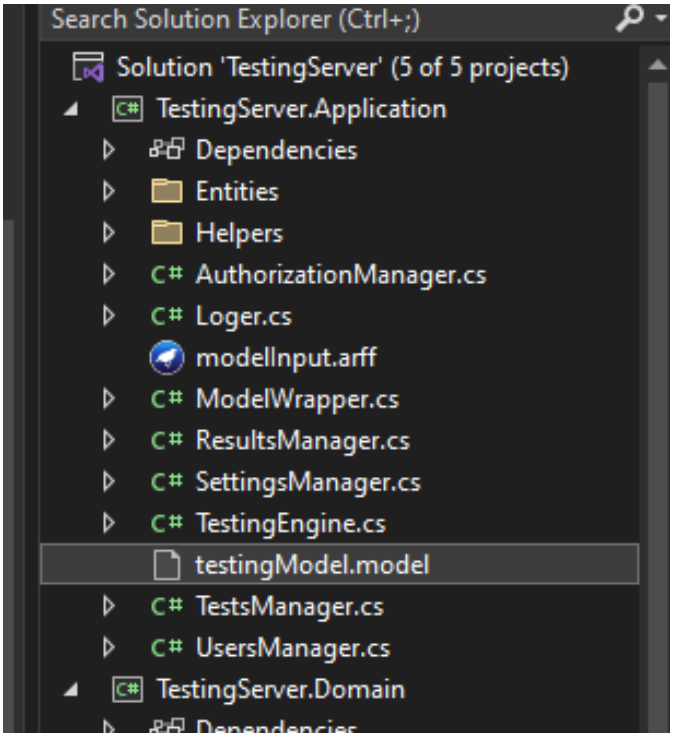


Рисунок 4.11 – Підключення експортованої моделі до проекту

Додатково було розроблено клас-обгортку для використання моделі. Як тільки користувач виконає 30% завдань з тестового набору, не важливо успішно чи ні, в процес тестування включається модель. Після кожного наступного завдання вона намагається класифікувати користувача відповідно класам 1 – 100, кожен з яких відповідає відсотку від максимально можливої кількості балів в тестовому наборі. Як тільки модель класифікує користувача з ймовірністю 80% і більше, процес тестування завершується і користувач отримує оцінку відповідно до класу, до якого модель його віднесла. Результат одного запиту на класифікацію до моделі у режимі відладки зображено на рисунку 4.12

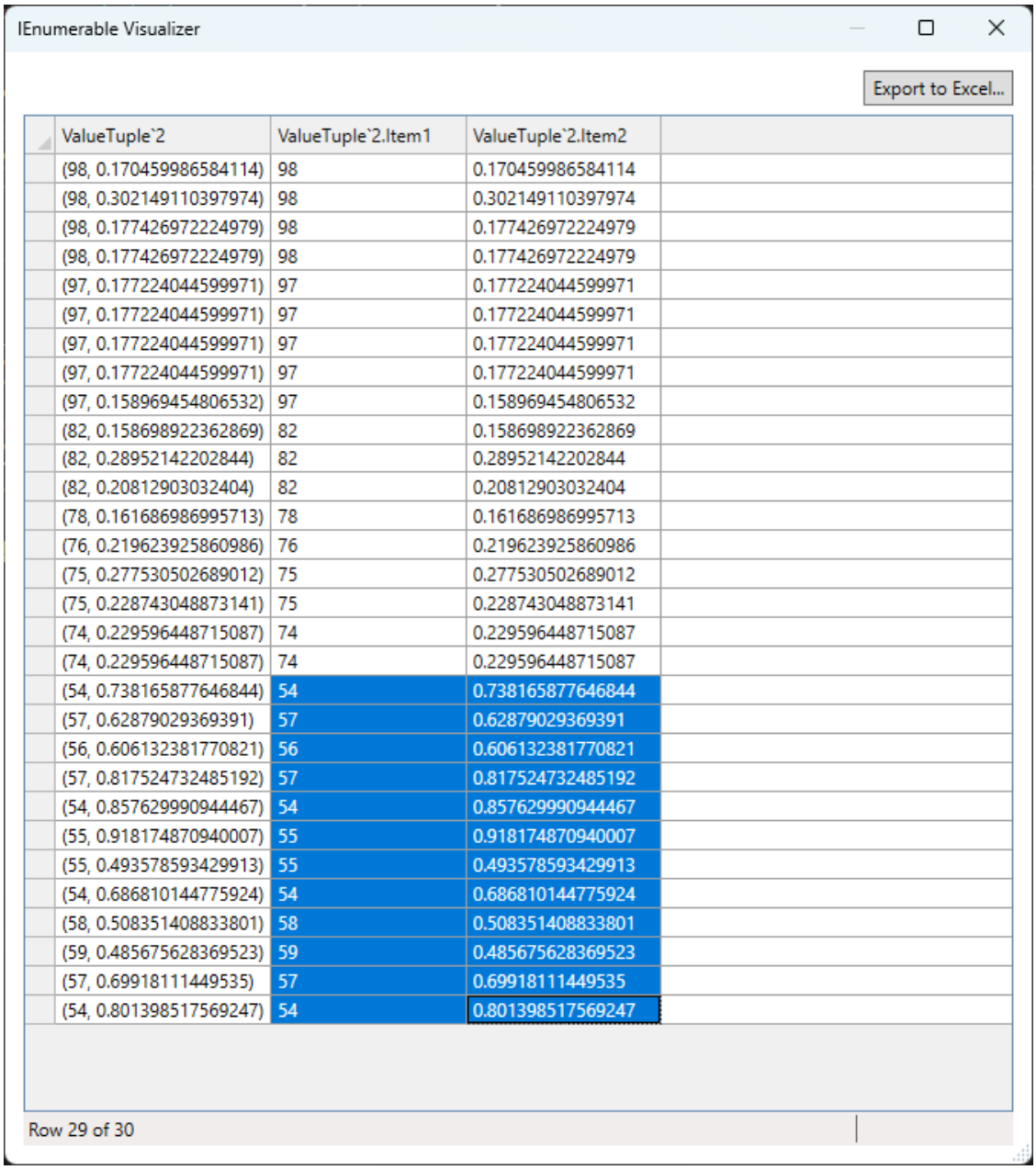


Рисунок 4.12 – Збільшення ймовірності класифікації під час проходження тестування

Як видно з даних класифікації, починаючи із завдання №19 модель змогла класифікувати користувача із вірогідністю 73% до класу 54. Вхідна комбінація відповідей відповідала класу 56, що є дуже наближеним. Загалом після завдання №19 ймовірність класифікації різко зросла від 15-20% до 50-85%, що вказує на те, що модель змогла розпізнати шаблон відповідей нових, невідомих їх раніше даних і успішно класифікувати користувача. Відповідно до імплементації, як тільки модель класифікує користувача з ймовірністю 70% або більше, процес тестування завершується і користувач отримує оцінку відповідно до свого класу.

4.6 Висновки

В ході варіантного аналізу та обґрунтування вибору способів реалізації програмного засобу було виконано низку завдань щодо вибору платформи проектування і власне реалізації програмного продукту.

Був проведений аналіз і обґрунтування вибору інструментальних засобів для проектування і розробки програмного забезпечення. в результаті чого було обрано інструментальний засіб Visual Paradigm для проектування системи і Visual Studio для реалізації програмного забезпечення.

Вибір мови програмування C# був обґрунтований його потужним функціоналом, широкою підтримкою та доступністю для розробників.

Була здійснена реалізація програмного забезпечення на основі згенерованого коду, з використанням інтегрованого середовища розробки Visual Studio.

Крім того, було інтегровано навчену модель штучного інтелекту до серверної частини системи та задіяно її в процесі тестування.

5 ТЕСТУВАННЯ ПРОГРАМНОГО ДОДАТКУ

5.1 Аналіз методів і засобів тестування

Сьогодні існує величезна кількість різних методів тестування програмного забезпечення. Серед найбільш відомих можна виділити наступні:

- Модульне тестування (Unit Testing) – тестування мінімально можливих окремих компонентів, функцій, програми. Таке тестування не повинно перевіряти будь-які інтеграційні сценарії, інфраструктурні проблеми, а лише код розробника.

- Інтеграційне тестування (Integration Testing) – відрізняється від модульного тестування тим, що воно перевіряє здатність двох або більше програмних компонентів працювати разом, також відому як їх «інтеграція». Ці тести працюють на більш широкому спектрі тестованої системи, тоді як модульні тести зосереджуються на окремих компонентах. Часто інтеграційні тести включають питання інфраструктури.

- Навантажувальне тестування (Load Testing) – має на меті визначити, чи здатна система витримати певне навантаження. Наприклад, кількість одночасних користувачів, які використовують програму, і здатність програми оперативно обробляти взаємодії [47, 48].

Тестування .NET додатків можна проводити за допомогою різних інструментів і бібліотек. Найпопулярніші інструменти для тестування .NET:

- Visual Studio: має вбудовану підтримку для тестування .NET додатків. Ви можете створювати і виконувати одиниці тестування за допомогою фреймворку MSTest, NUnit або xUnit;

- NUnit – це популярний фреймворк для юніт тестування, який підтримує тестування .NET додатків;

- xUnit – інший популярний фреймворк для одиниць тестування в .NET. Він спроектований для спрощення структури та організації тестів;

- Moq – це бібліотека для створення об'єктів-заглушок (mock objects) в .NET, які допомагають вам виконувати модульне тестування залежностей

вашого коду;

- Selenium – може бути корисним інструментом для автоматизованого тестування додатків з веб-інтерфейсом;

- SpecFlow – це інструмент для Behavior-Driven Development (BDD), який допомагає описувати поведінку додатку за допомогою природної мови та автоматизовувати тестування на основі цих описів;

- Postman – зручний інструмент для тестування веб-служб або API на платформі .NET;

- Visual Studio Test – вбудований менеджер для написання та запуску тестів в середовищі розробки Visual Studio [49].

Ці інструменти дозволяють проводити різні типи тестування, включаючи модульне тестування, інтеграційне тестування, автоматизоване тестування і багато інших. Вибір конкретного інструмента може залежати від типу додатка і потреб тестування [50].

5.2 Етапи тестування програмного додатку

Тестування проводилось на створеному заздалегідь тесті, що містить 320 завдань різноманітної складності. Для адаптивного тестування необхідним є великий банк завдань, оскільки тестований може багато разів повертатись до одного й того ж рівня складності тому потрібно завжди мати в наявності різнорівневі завдання. Основне навантаження на сервер припадає під час передавання картинок великих розмірів одночасно багатьом клієнтам, але це обмеження не стосується розробленої системи, а скоріше залежить від швидкості передачі даних до мережі.

Для тестування системи було обрано підхід інтеграційного тестування, оскільки додаток розподілений то тестувати необхідно повні бізнес сценарії, починаючи з клієнтського додатку і закінчуючи перевіркою даних в базі даних.

Тестування клієнтського додатку зображено на рисунку 5.1.

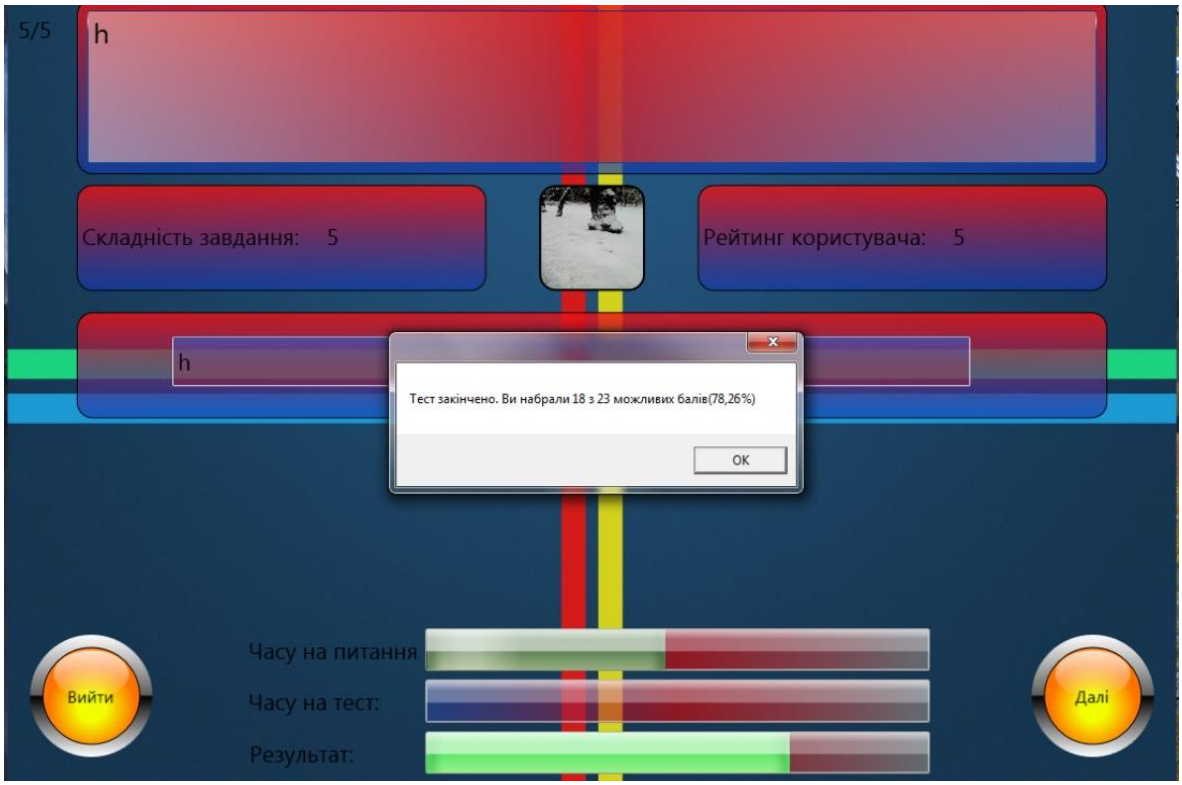
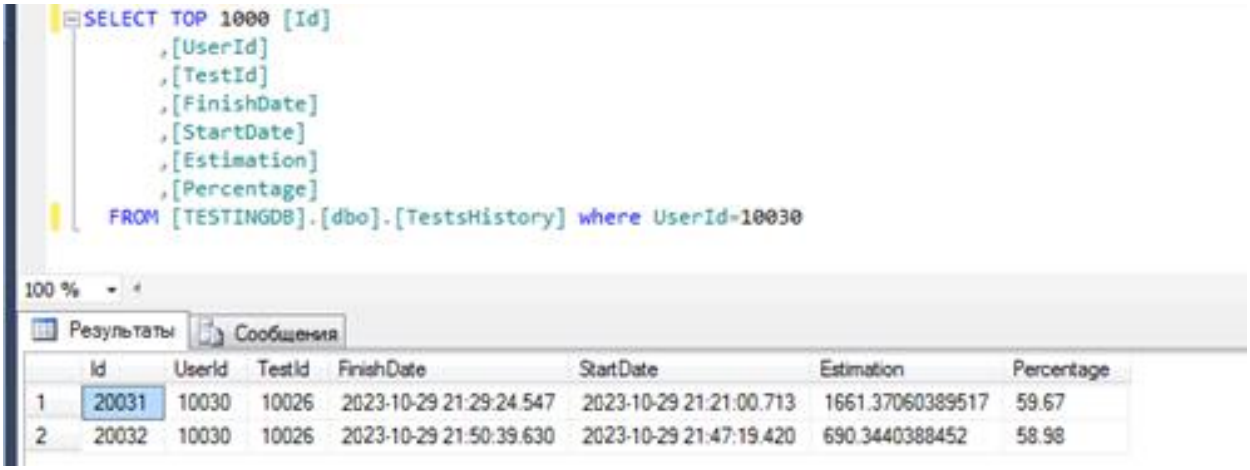


Рисунок 5.1 – Тестування сценарію «Проходження тесту»

Сервер зберігає історію проходження тестування всіма користувачами. Запис історії тестування для пройденого тесту зображено на рисунку 5.2.



```
SELECT TOP 1000 [Id]
      ,[UserId]
      ,[TestId]
      ,[FinishDate]
      ,[StartDate]
      ,[Estimation]
      ,[Percentage]
  FROM [TESTINGDB].[dbo].[TestsHistory] where UserId=10030
```

100 %

Результаты | Сообщения

	Id	UserId	TestId	FinishDate	StartDate	Estimation	Percentage
1	20031	10030	10026	2023-10-29 21:29:24.547	2023-10-29 21:21:00.713	1661.37060389517	59.67
2	20032	10030	10026	2023-10-29 21:50:39.630	2023-10-29 21:47:19.420	690.3440388452	58.98

Рисунок 5.2 – Результати тестування для даного користувача

Під час тестування, складність завдань підлаштовується під рівень підготовки користувача. Діаграму адаптивності системи, яка відображає складність кожного наступного завдання під час останнього тестування зображено на рисунку 5.3.

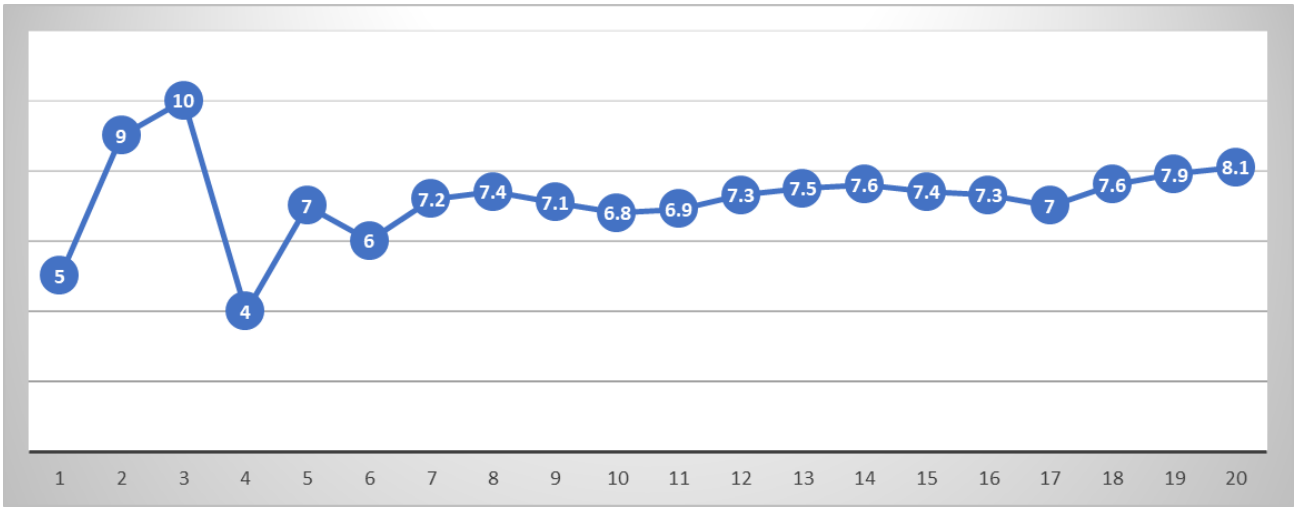


Рисунок 5.3 – Діаграма адаптивності системи

Найважливішим модулем системи є модель класифікації яка оцінює користувачів. Тестування цього функціоналу було проведено за допомогою порівняння двох тестових наборів. Перший тестовий набір згенерований у Microsoft Excel, в якому у випадковому порядку було згенеровано 100 сценаріїв проходження тестовоно завдання без використання моделі оцінки. Кожен сценарій містить випадкові відповіді на 20 завдань та результуючу оцінку, яка обраховується як відсоток від максимально можливої оцінки за даний тест.

Другий тестовий набір було скопійовано з першого, з видаленням результуючої оцінки. За допомогою Unit тесту модель викликалась для кожного сценарію з другого набору, та зберігався результат її оцінки, ймовірність класифікації та кількість завдань, які знадобились моделі для здійснення прийняття рішення. Фрагмент тестового набору зображено на рисунку 5.4.

№ завдання																
15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	Результат
7.235009	7.81381	0	7.968539	8.845078	9.287332	9.473078	0	0	6.871945	7.49042	0	0	0	0	0	39
0	0	0	5.177354	5.746863	6.379018	0	0	5.85231	6.788679	7.060227	0	0	5.970471	0	6.193666	32
0	4.39152	4.435435	0	0	0	0	0	2.55871	2.840169	3.379801	0	0	3.48463	3.693708	0	19
0	0	0	0	0	2.848837	2.877326	0	0	0	0	0	1.632986	1.632986	0	1.542264	13
0	7.588078	0	0	6.321566	7.14337	8.286309	0	0	0	6.943446	8.332136	9.581956	10	10	0	44
2.361602	2.763074	0	2.925608	3.51073	0	3.123954	0	2.670047	0	0	2.149796	0	2.056327	2.159143	0	15
0	4.382497	0	0	0	3.400244	3.672263	0	0	3.416802	0	0	2.789119	0	0	2.632462	18
0	1.500616	1.755721	1.808392	0	1.616351	0	1.56013	0	0	0	1.427596	0	1.639615	0	1.393673	6
5.828948	0	0	4.755122	0	4.711893	4.759012	5.710815	0	0	0	5.640927	6.487066	0	0	5.795787	28
2.511572	0	0	0	2.057991	0	2.397659	2.661401	3.167068	3.325421	0	3.537039	3.961483	4.516091	5.328987	5.542147	21
6.059782	0	5.670622	6.407802	0	0	4.958137	5.156463	5.981497	0	6.253383	7.128857	8.554628	10	0	9.52381	46
8.056169	8.056169	0	8.056169	9.184033	0	0	0	7.903089	8.772429	0	0	0	0	7.792987	8.494356	42
0	5.510693	0	0	4.87672	0	0	0	3.747669	0	3.747669	0	0	3.200271	0	0	27
2.821798	2.906452	0	0	2.572887	2.650074	0	0	2.278502	0	0	0	0	0	0	1.497839	11

Рисунок 5.4 – Фрагмент тестового набору

В кожному рядку таблиці міститься кількість балів, які користувач отримав за відповідь на завдання: певне число, якщо правильно, та 0 – якщо неправильно. Кількість балів є співрозмірною до складності завдання, яке користувач вирішив. В колонці «Результат» міститься відсоток від максимально можливого результату, який користувач міг би отримати, відповівши правильно на всі завдання.

При виклику класифікаційної моделі для таких самих вхідних даних отримано іншу таблицю з результатами. Результат порівняння тестування без класифікаційної моделі та з нею наведено на рисунку 5.5.

	A	B	C	D	E	F
1	**Результат**		**Результат**	**Ймовірність**	**Кількість завдань**	**Похибка**
2	39		35	76%	22	-10%
3	32		32	75%	22	0%
4	19		19	77%	23	0%
5	13		11	87%	26	-14%
6	44		46	84%	25	5%
7	15		16	84%	25	11%
8	18		18	81%	24	5%
9	6		6	73%	21	0%
10	28		28	80%	24	0%
11	21		19	84%	25	-5%
12	46		36	75%	22	-20%
13	42		42	93%	27	0%
14	27		27	83%	24	0%
15	11		11	77%	23	0%
16	54		55	88%	26	2%
17	18		15	70%	21	-13%
18	15		15	73%	21	0%
19	21		24	73%	21	18%
20	40		40	77%	23	0%
21	50		57	78%	23	14%
22	25		25	86%	25	1%
23	30		30	75%	22	3%
24	24		24	82%	24	0%
25	39		39	88%	26	0%
26	32		29	88%	26	-9%
27	48		45	79%	23	-5%
28	24		24	83%	24	0%
29	8		8	86%	25	6%
30	38		34	80%	24	-8%
31	36		42	87%	26	19%
32	15		15	93%	27	3%

Рисунок 5.5 – Порівняння результатів тестування з деревом рішень.

Отже, порівнявши результати тестування, можна зробити наступні висновки:

- середнє відхилення результату, передбаченого моделлю штучного інтелекту від еталонного значення склало 6%. Похибка коливалась від 0% до 21%;

- середня ймовірність класифікації моделі склала 83%, коливалась від 72% до 96%;

- кількість пройдених завдань в кожному сценарії в середньому склала 24.5 завдання, коливалась від 22 до 28, тоді як у тестуванні без моделі класифікації кожне тестування зобов’язувало пройти 30 завдань;

- в середньому кожне тестування зайняло на 18.3% менше часу (через зменшену кількість необхідних завдань).

5.3 Формування інструкції користувача

Головне вікно клієнтського додатку містить 3 основні кнопки: «авторизація», «тестування», «реєстрація».

Перед початком роботи, потрібно встановити з’єднання з сервером. Для цього слугує кнопка «Зв’язок». Якщо зв'язок успішно встановлено, то основні функції будуть розблоковані. Користувач може зареєструватись, авторизуватись в мережі, якщо його обліковий запис вже існує, та почати тестування.

Вікно тестування містить поле для зображення. Якщо тестове завдання передбачає використання зображення, то клацнувши на цьому полі, зображення відкриється на повний екран для перегляду.

Завдання, які містять коло варіантів відповідей яких є елементи CheckBox можуть мати від 0 до 4 правильних відповідей. Завдання з елементами RadioButton можуть містити лише 1 правильну відповідь. Завдання з полем вводу потребують безпосереднього вводу відповіді.

При натисненні кнопки «Вийти», всі завдання, які залишились невиконаними, будуть автоматично зараховані як неправильні. Коли таймер часу

на запитання досягає нуля, відбувається зміна завдання. Якщо при цьому було обрано правильну відповідь, вона автоматично зарахується.

Під таймерами відображається рядок результату, що відображає результат у відсотковому відношенні, зароблених балів до втрачених.

5.4 Висновки

Було розглянуто основні методи та засоби для тестування програмного забезпечення. Проведено інтеграційне тестування застосунку, усунуто виявлені несправності та складено інструкцію користувача для подальшого використання розробки.

Проведено модульне тестування розробленої моделі дерева рішень на основі підготовлених наборів випадкових тестових даних, виконано порівняння результатів тестування на однакових даних без використання класифікаційної моделі та з нею.

Результат тестування підтвердив гіпотезу дослідження в тому, що вдалось досягти збільшення швидкості проведення тестування знань та економію ресурсів для його проведення на 18.3% в порівнянні з аналогічними розробками без використання моделі класифікації.

6 ЕКОНОМІЧНА ЧАСТИНА

6.1 Оцінювання комерційного потенціалу розробки

Метою проведення комерційного аудиту є оцінювання комерційного потенціалу впровадження методів та засобів, розробленої системи адаптивного тестування знань.

Для проведення технологічного аудиту було залучено 3-х незалежних експертів Вінницького національного технічного університету к.т.н., доцента Майданюка В. П., к.т.н., доцента Ракитянську Г. Б., к.т.н., доцента Рейди О.М з кафедри програмного забезпечення.

Аудит науково-технічної розробки та її комерційного потенціалу проведено за допомогою таблиці 6.1, застосовуючи п’ятибальну шкалу оцінювання за 12-ма критеріями оцінки [51].

Таблиця 6.1 – Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
#	0	1	2	3	4
1	2	3	4	5	6
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів

Продовження таблиці 6.1.

1	2	3	4	5	6
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
11	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Продовження таблиці 6.1.

12	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві

В таблиці 6.2 наведено результати оцінювання науково-технічного рівня і комерційного потенціалу розробки.

Таблиця 6.2 – Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали, посада експерта		
	1. Майданюк В. П.	2. Ракитянська Г. Б.	3. Рейда О. М.
	Бали, виставлені експертами:		
1	2	4	1
2	3	2	3
3	4	3	2
4	3	1	3
5	1	3	3
6	2	3	3
7	1	4	4
8	2	3	2
9	3	3	2
10	3	3	2
11	3	3	4
12	4	1	1
Сума балів	$СБ_1$=31	$СБ_2$=33	$СБ_3$=30
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_1^3 СБ_i}{3} = \frac{31 + 33 + 30}{3} = 31.3$		

В таблиці 6.3 наведено шкалу оцінки комерційного потенціалу розробки.

Таблиця 6.3 – Рівні комерційного потенціалу розробки

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

За результатами розрахунків, наведених в таблиці 6.2 та шкалою оцінки наведеної в таблиці 6.3 можна зробити висновок щодо рівня комерційного потенціалу розробки. Середньоарифметична сума балів, виставлених експертами склала 31.3, що відповідає рівню «вище середнього».

Такий рівень комерційного потенціалу досягнуто за рахунок суттєвого зменшення витрат ресурсів і часу на проведення тестування знань. Особливо відчутно це при проведенні тестування великої кількості людей, що характерно великим організаціям, як державним (в більшості це освітні установи, сервісні центри МВС), так і приватні (сертифікаційні організації). Передусім розробка дозволяє економити людські ресурси, необхідні для організації і проведення тестування, амортизаційні відрахування обладнання, та витрати електроенергії що споживається комп'ютерним обладнанням.

Така економія забезпечується використанням моделі машинного навчання, яка прогнозує результат тестування та гарантовано зменшує кількість тестових завдань, які тестований зобов'язаний вирішити в процесі тестування.

6.2 Прогнозування витрат на виконання науково-дослідної роботи

Витрати, пов'язані з проведенням науково-дослідної роботи можна розрахувати за наступними статтями:

- Витрати на оплату праці;
- Витрати на інструментальне та інфраструктурне програмне забезпечення;
- Амортизаційні відрахування;
- Енергія для науково-виробничих цілей.

6.2.1 Основна заробітна плата

Заробітна плата кожного із залучених осіб визначається за формулою 6.1

$$З_0 = \sum_{i=1}^{K} \frac{M_{ni} * t_i}{T_p} \text{ (грн)} \qquad (6.1)$$

Де k – кількість посад працівників, залучених до процесу дослідження і розробки;

M_{ni} – місячний посадовий оклад конкретного розробника (інженера, дослідника, науковця тощо), грн.;

$T_р$– число робочих днів в місяці; приблизно $T_р = 22$;

t - кількість робочих днів роботи працівника.

Для проектування і розробки системи адаптивного тестування знань було залучено наступних робітників: Solution Architect, Software Engineer та Quality Assurance Engineer. Посадові оклади, число днів роботи та витрати на компенсацію наведено в таблиці 6.4

Таблиця 6.4 - Компенсація спеціаліста в дослідницькій установі

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату грн.
Project Manager	40000	1818	24	43636
Solution Architect	44000	2000	10	20000
Software Engineer	38000	1727	24	41448
Quality Assurance Engineer	23000	1045	12	12540
Всього				117624

6.2.2 Розрахунок додаткової заробітної плати робітників

Додаткова заробітна плата $З_д$ всіх робітників, які приймали участь в розробці нового технічного рішення розраховується за формулою 6.2 як 10 - 15 % від основної заробітної плати робітників як премія.

На даному підприємстві додаткова заробітна плата нараховується в розмірі 10% від основної заробітної плати.

$$З_д = (З_о + З_р) * \frac{Н_{доп}}{100\%} \quad (6.2)$$

$$З_д = 0{,}1 * 117624 = 11762 \text{ грн}$$

5.2.3 Нарахування на заробітну плату

Нарахування на заробітну плату $Н_{ЗП}$ робітників, які брали участь у виконанні роботи, розраховуються за формулою (6.3):

$$З_н = (З_о + З_р + З_д) * \frac{Н_{ЗП}}{100} \text{ (грн)} \quad (6.3)$$

де $З_о$ – основна заробітна плата розробників, грн.;

$З_д$ – додаткова заробітна плата всіх розробників та робітників, грн.;

$Н_{ЗП}$ – ставка єдиного внеску на загальнообов'язкове державне соціальне страхування, % .

Основна ставка єдиного внеску на загальнообов'язкове державне соціальне страхування на 2023 рік – 22 %, тоді:

$$З_н = (117624 + 11762) * 0.22 = 28464(\text{грн})$$

6.2.4 Витрати на матеріали та комплектуючі вироби

Дана стаття витрат включає витрати на матеріали, пристрої, засоби, які використовують при виготовленні одиниці продукції. Розраховуються, згідно їх номенклатури, за формулою 6.4:

$$К = \sum_{i=1}^{n} Н_i \cdot Ц_i \cdot К_i, \quad (6.4)$$

де $Н_i$ – кількість комплектуючих *i*-го виду, шт.;

$Ц_i$ – покупна ціна комплектуючих *i*-го найменування, грн.;

$К_i$ – коефіцієнт транспортних витрат (1,1…1,15).

Інформацію про використані матеріали та комплектуючі наведено у таблиці 6.5.

Таблиця 6.5 – Матеріали, що використані на розробку

Найменування матеріалу	Ціна за одиницю, грн.	Витрачено	Вартість витраченого матеріалу, грн.
Папір офісний А4 Zoom білий 250 аркушів	205	1	205
Тонер ColorWay для принтера HP LaserJet 1018	129	1	129
Набір маркерів кольорових Stanger 8 шт.	180	1	180
Всього			514
З врахуванням коефіцієнта транспортування			565

Узагальнені витрати на доставку матеріалів та комплектуючих закладаються у вигляді коефіцієнту транспортних витрат, в даному випадку прийнято значення – 1.1.

6.2.5 Витрати на програмне забезпечення

До даної статті входять витрати на програмне забезпечення, необхідне для проектування та розробки адаптивної системи тестування знань. Балансову вартість програмного забезпечення розраховують за формулою 6.5:

$$В_{прг} = \sum_{i=1}^{k} Ц_{іпрг} \cdot С_{пргі} \cdot К_і, \quad (6.5)$$

де $Ц_{іпрг}$ – ціна придбання/використання одиниці програмного засобу цього виду, грн;

$С_{пргі}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

$К_і$ – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо ($К_і$ = 1,10…1,12).

k – кількість найменувань програмних засобів.

Отримані результати наведено в таблиці 6.6.

Таблиця 6.6 – Витрати на використання програмних

Найменування устаткування	Час використання, місяців	Ціна за місяць, грн	Вартість, грн
Підписка Visual Paradigm Enterprise	1	3660	3660
Підписка Microsoft Visual Studio Professional 22	2	1647	3294
Підписка MS SQL Server Standard	2	2671	5342
Всього			12296

6.2.6 Амортизація обладнання

До даної статті включаються амортизаційні відрахування по кожному виду обладнання, устаткування яке використовувалось для проектування та розробки

системи адаптивного тестування знань.

$$А_{обл} = \frac{Ц_б}{Т_в} * \frac{t_{вик}}{12} \quad (6.6)$$

де $Ц_б$ – балансова вартість даного виду обладнання (приміщень), грн.;

$t_{вик}$ – час користування;

$Т_в$ – термін використання обладнання (приміщень), цілі місяці.

Для розробки та хостингу продукту використовувався персональний комп'ютер вартістю 112000 грн. Для тестування використовувався ноутбук вартістю 23000 грн. Амортизаційні відрахування наведено в таблиці 6.7.

Таблиця 6.7 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Комп'ютер	112000	5	2	3733
Ноутбук	23000	4	2	958
Офісне приміщення	2000000	30	2	11111
Всього				15802

6.2.7 Енергія для науково-виробничих цілей

До даної статті відносяться витрати на всі види палива й енергії, що безпосередньо використовуються з технологічною метою на проведення досліджень.

$$В_е = \sum_{i=1}^{n} \frac{W_{yt} \cdot t_i \cdot Ц_е \cdot К_{впі}}{\eta_i} \quad (6.7)$$

де W_{yt} – встановлена потужність обладнання на певному етапі розробки, кВт;

t_i – тривалість роботи обладнання, год;

$Ц_е$ – вартість 1 кВт-години електроенергії, грн;

$К_{впі}$ – коефіцієнт, що враховує використання потужності, $К_{впі} < 1$;

$\eta_і$ – коефіцієнт корисної дії обладнання, $\eta_і < 1$.

Вартість електроенергії становить 7.6 грн за кВт в 2023 році. При розробці системи використовувалось 2 комп'ютери з сумарною потужністю 0.9 кВт, освітлення, вентиляція та кондиціювання має сумарну потужність 0.5 кВт. Сумарні витрати на електроенергію становлять:

$$В_е = \frac{(0.9 + 0.5) \cdot 780 \cdot 7.6 \cdot 0.45}{0.76} = 4914$$

6.2.8 Витрати на роботи, які виконують сторонні підприємства, установи і організації

Стаття включає витрати на відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих розробленням досліджень, відрядження, пов'язані з проведенням випробувань машин та приладів, а також витрати на відрядження на наукові з'їзди, конференції, наради, пов'язані з виконанням конкретних досліджень. Витрати за статтею «Службові відрядження» розраховуються як 20…25% від суми основної заробітної плати дослідників та робітників за формулою 6.8:

$$В_{св} = (З_о + З_р) \cdot \frac{Н_{св}}{100} \quad (6.8)$$

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

$$В_{св} = 117624 * 0.2 = 23524 \text{ грн}$$

6.2.9 Витрати на роботи, які виконують сторонні підприємства, установи і організації

Дана стаття витрат включає витрати на проведення досліджень, що не можуть бути виконані штатними працівниками або наявним обладнанням організації, а виконуються на договірній основі іншими підприємствами, установами і організаціями незалежно від форм власності та позаштатними працівниками. Такі витрати розраховуються як 30…45% від суми основної

заробітної плати дослідників та робітників за формулою 5.9.

$$В_{сп} = \left(З_о + З_р\right) \cdot \frac{Н_{сп}}{100} \quad (6.9)$$

де $Н_{пв}$ – норма нарахування за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації».

$$В_{сп} = 117624 * 0.30 = 35287 \text{ грн}$$

6.2.10 Інші витрати

До статті «Інші витрати» належать витрати, які не знайшли відображення у попередніх статтях витрат і можуть бути віднесені безпосередньо на собівартість розробки за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуються як 50…100% від суми основної заробітної плати робітників за формулою 5.10

$$І_в = \left(З_о + З_р\right) \cdot \frac{Н_{ів}}{100} \quad (6.10)$$

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

$$І_в = 117624 * 0.5 = 58812 \text{ грн}$$

6.2.8 Загальновиробничі витрати

Дана стаття витрат охоплює витрати на управління організацією, оплату службових відряджень, витрати на утримання, ремонт та експлуатацію основних засобів, витрати на опалення, освітлення, водопостачання, охорону праці тощо. Накладні (загальновиробничі) витрати $B_{нзв}$ можна прийняти як 120% від суми основної заробітної плати розробників та робітників, які виконували дану МКНР, тобто:

$$В_{нзв} = \left(З_о + З_р\right) \cdot \frac{Н_{нзв}}{100}, \quad (6.8)$$

де $Н_{нзв}$ – норма нарахування за статтею «Загальновиробничі витрати».

$$В_{нзв} = 117624 * 1.2 = 141148 \text{ грн}$$

6.2.9 Загальні витрати

Сума всіх статей витрат дає в результаті витрати на проведення дослідження та розробку адаптивної системи тестування знань і розраховується за формулою:

$$В = З_о + З_р + З_{дод} + З_н + М + К_в + В_{спец} + В_{прг} + А_{обл} + В_е + В_{св} + В_{сп} + I_в + В_{нзв} \quad (6.9)$$

$$В = 117624 + 11762 + 17904 + 12296 + 15802 + 4914 + 58812 + 141148 = 414911 \text{грн}$$

Загальні витрати ЗВ на завершення роботи та оформлення її результатів розраховуються за формулою:

$$ЗВ = \frac{В}{\eta}, \quad (6.10)$$

де η – коефіцієнт, який характеризує стадію виконання даної НДР.

Оскільки, робота знаходиться на стадії дослідного зразка, то коефіцієнт β = 0.5.

Звідси:

$$ЗВ = \frac{414911}{0.5} = 829822 \text{грн.}$$

6.3 Розрахунок економічної ефективності науково-технічної розробки

Економічна ефективність дозволяє спрогнозувати чистий прибуток, який може бути отриманий від впровадження розробленої системи.

При розрахунку економічної ефективності потрібно обов'язково враховувати зміну вартості грошей у часі, оскільки від вкладення інвестицій до отримання прибутку минає чимало часу [52].

Для розрахунку збільшення чистого прибутку підприємства $\Delta П_i$, для кожного із років, протягом яких очікується отримання позитивних результатів від

впровадження розробки використовується формула формулою 6.11:

$$\Delta П_i = \sum_1^n \left(\Delta Ц_o \cdot N + Ц_o \cdot \Delta N\right)_i \cdot \lambda \cdot \rho \cdot \left(1 - \frac{\upsilon}{100}\right) \quad (6.11)$$

де $\Delta Ц_0$ – покращення основного оціночного показника від впровадження результатів розробки у даному році.

N – основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження нової розробки;

ΔN – покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки:

Цо – основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів розробки;

n – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки:

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

ρ – коефіцієнт, який враховує рентабельність продукту. р = 0,25;

ϑ – ставка податку на прибуток. У 2023 році – 18%.

Впровадження результатів розробки дозволяє швидше виконувати тестування знань. Припустимо, сертифікаційний центр проводить тестування знань в певній сфері знань, отримуючи прибуток 2000 грн за одного тестованого користувача. Один сеанс тестування займає приблизно 1 годину, враховуючи що тестовий набір має близько 50 завдань. Маючи приміщення на 20 робочих місць, працюючи 5 днів на тиждень, компанія може обслуговувати приблизно 100 користувачів на добу, або 2000 користувачів на місяць, та, відповідно 24000 користувачів на рік. Враховуючи, що покращена система скорочує час тестування одного користувача в середньому на 50%, компанія спроможна обслуговувати вдвічі більше користувачів, витрачаючи однакові ресурси. Важливо, щоб маркетинговий відділ забезпечив достатню зацікавленість аудиторії до продукту компанії, тому будемо вважати, що кількість користувачів

буде збільшуватись поступово на 15% щорічно. Крім того, вартість послуг компанії, може поетапно збільшуватись на 10% разом із покращенням якості надаваної послугу користувачам.

$$\Delta П_1 = (200 \cdot 24000 + (2000 + 200) \cdot 3600) \cdot 0.833 \cdot 0.25 \cdot \left(1 - \frac{18}{100}\right)$$
$$= (4800000 + 7920000) \cdot 0.170765 = 2172130 = 2.17 \text{ млн}$$

$$\Delta П_2 = (200 \cdot 24000 + (2000 + 400) \cdot 7200) \cdot 0{,}833 \cdot 0{,}25 \cdot \left(1 - \frac{18}{100}\right)$$
$$= (4.800000 + 17.280000) \cdot 0.170765 = 3770491 = 3.77 \text{ млн}$$

$$\Delta П_3 = (200 \cdot 24000 + (2000 + 600) \cdot 10800) \cdot 0.833 \cdot 0.25 \cdot \left(1 - \frac{18}{100}\right)$$
$$= (4800000 + 17280000) \cdot 0.170765 = 5614753 = 5.61 \text{ млн}$$

Далі за формулою 6.12 розраховують приведену вартість збільшення всіх чистих прибутків ПП, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t} \qquad (6.12)$$

де ΔПі – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

Т – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ =0,05…0,15. В Україні рівень інфляції за підсумком 2023 року склав 10.6%, прогнозований рівень на 2024 рік – 8.5% ;

t – період часу (в роках) від моменту початку впровадження розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$$ПП = \frac{2172130}{1 + 0.1} + \frac{3770491}{(1 + 0.085)^2} + \frac{5614753}{(1 + 0.085)^3}$$
$$= 1974663 + 3202863 + 4395835 = 9573361$$

За формулою 5.13 необхідно розрахувати величину початкових інвестицій PV , які потенційний інвестор має вкласти для впровадження і комерціалізації розробки.

$$PV = k_{\text{інв}} \cdot \text{ЗВ} \quad (6.13)$$

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

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

$$PV = 4 \cdot 829822 = 3319288 \text{ грн}$$

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

$$E_{\text{абс}} = \text{ПП} - PV \quad (6.14)$$

де ПП – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, грн;

PV – теперішня вартість початкових інвестицій, грн.

$$E_{\text{абс}} = 9573361 - 3319288 = 6254073 \text{ грн}$$

Оскільки величина економічного ефекту має велике додатне значення, це свідчить про потенційну зацікавленість інвесторів у впровадженні та комерціалізацію розробки.

6.4 Розрахунок ефективності вкладених інвестицій та періоду їх окупності

Для остаточного прийняття рішення про впровадження розробки та

виведення її на ринок необхідно розрахувати внутрішню економічну дохідність $E_в$ або показник внутрішньої норми дохідності (IRR, Internal Rate of Return) вкладених інвестицій та порівняти її з так званою бар'єрною ставкою дисконтування, яка визначає ту мінімальну внутрішню економічну дохідність, нижче якої інвестиції в будь-яку розробку вкладати буде економічно недоцільно [34, 35].

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $E_в$. Для цього користуються формулою 6.15:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1, \tag{6.15}$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, грн; PV – теперішня вартість початкових інвестицій, грн;

$T_ж$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, роки.

$$E_в = \sqrt[3]{1 + \frac{6254073}{3319288}} - 1 = \sqrt[3]{2.884} - 1 = 1.4234157 - 1 = 0.42 = 42\%$$

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою 6.16:

$$\tau = d + f, \tag{6.16}$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2023 році в Україні d= (0,14…0,2);

f – показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,1).

$$\tau_{min} = 0.17 + 0.07 = 0.24$$

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

Далі розраховується період окупності інвестицій, вкладених у реалізацію проекту за формулою 6.17.

$$T_{ок} = \frac{1}{E_в} \tag{6.17}$$

де Ев – внутрішня економічна дохідність вкладених інвестицій.

$$\mathrm{T_{ок}} = \frac{1}{0.42} = 2.38 \text{ роки} = 28.6 \text{міс} = 2 \text{ роки та } 5 \text{ міс}$$

Так як $\mathrm{T_{ок}} \leq 3$ років, то це свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження цієї розробки та виведення її на ринок.

6.5 Висновки

В даному розділі було проведено оцінку комерційного потенціалу розробки програмного забезпечення для адаптивного тестування знань. Економічний потенціал склав значення, яке є вище середнього.

Також було зпрогнозовано витрати на проектування та реалізацію системи за необхідними статтями витрат, які склали 414911 грн. Загальна величина витрат склала 829822 грн.

Обрахунок економічної ефективності та терміну окупності вкладених інвестицій показали, що розробка є економічно доцільною та привабливою для потенційних інвесторів. Термін окупності інвестицій складає 2 роки та 5 місяців а прогнозований прибуток за 3 роки 9573361 грн

ВИСНОВКИ

В даній роботі було розглянуто необхідність впровадження систем адаптивного тестування знань у різноманітні сфери людського життя, де необхідна оцінка знань. В результаті аналізу існуючих рішень було сформовано основні проблеми сучасних систем адаптивного тестування знань, а також актуальність вирішення даних проблем.

Під час виконання роботи було розроблено метод оцінки знань користувача та завершення тестування за рахунок підготовки та навчання класифікаційної моделі алгоритму дерева рішень. Дану модель було інтегровано в систему для забезпечення кращої швидкодії та економії ресурсів, необхідних для проведення тестування. Результат перевірки даного методу підтвердив гіпотезу дослідження в тому, що вдалось досягти збільшення швидкості проведення тестування знань та економію ресурсів для його проведення на 18.3% в порівнянні з аналогічними розробками без використання моделі класифікації.

В рамках виконання роботи було покращено метод «адаптивності» системи тестування знань за рахунок нової формули передбачення складності наступних завдань. Даний метод забезпечує «підгонку» завдань до рівня користувача вже на перших декількох завданнях, що забезпечує ефективне та об’єктивне проведення оцінки знань. Крім того, було реалізовано метод самокалібрування завдань за складністю, що дозволяє усунути негативний вплив людського чиннику у процес укладання тестових завдань.

Для перевірки та тестування даних методів було спроектовано та розроблено систему адаптивного тестування знань у вигляді клієнт-серверного додатку. Тестування програмного засобу дозволило перевірити відповідність розроблених методів до поставленого завдання.

Було розраховано комерційний потенціал та економічну ефективність науково-технічної розробки та її економічну привабливість для потенційних інвесторів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Howard W. Computerized adaptive testing: A Primer (2nd Edition). Mahwah, NJ: ELawrence Erlbaum Associates, 2000, 361p.

2. Duanli Yan, Alina A. von Davier, Charles Lewis. Computerized Multistage Testing: Theory and Applications. Chapman and Hall/CRC, 2014, 546p.

3. Scott A. Crossley, Danielle S. McNamara, Scott A. Crossley, Danielle S. McNamara. Adaptive Educational Technologies for Literacy Instruction, Routledge, 2016, 310p.

4. Гороховський О. І., Ситніков Є. О. Використання нейромережевих технологій в адаптивному тестуванні знань. Четверта Міжнародна науково-практична конференція «Інформаційні технології та комп'ютерна інженерія», 2014. URL: https://epsi.vntu.edu.ua/download/tezy.pdf (дата звернення: 30.05.2023)

5. Ситніков Є. О. Розробка методів і засобів для систем адаптивного тестування знань. Міжнародна науково-практична Інтернет-конференція «Електронні інформаційні ресурси: створення, використання, доступ», 2023. URL: https://drive.google.com/file/d/1oVmxS3W_sEQPjes9S9AzWDaJxDxi6I0X/view (дата звернення: 08.12.2023).

6. Олійник М. М. Тест як інструмент кількісної діагностики рівня знань в сучасних технологіях навчання. Донецьк: Донецький національний університет, 2001. 83 с.

7. Robert L. Brennan. Educational Measurement (Ace Praeger Series on Higher Education). Rowman & Littlefield Publishers, 2006, 779p.

8. Drasgow F., Olson-Buchanan, J. B. Innovations in computerized assessment. Hillsdale, NJ: Erlbaum. 2006, 280p

9. Weiss D. J. New Horizons in Testing: Latent Trait Test Theory and Computerized Adaptive Testing. NY., Academic Press, 1983, 345p.

10. Smoline D. Some problems of computer-aided testing and "interview-like tests", Computers & Education, 2008. No. 1. P. 743–756.

11. Legg S. M. Computerized Adaptive Testing with Different Groups. Educational Measurement: Issues and Practice. 1992, № 11. P. 23–27.

12. Williams D. D. Online assessment, measurement, and evaluation: emerging practices. Hershey, PA: Information Science Pub, 2006. 343 p.

13. Hontangas P, Hontangas S., Ponsoda V., Olea J., Wise S. L. The choice of item difficulty in self adapted testing. European Journal of Psychological Assessment, 2000, № 16, С. 3–12.

14.Educational data mining: Prediction of students' academic performance using machine learning algorithms: URL: https://slejournal.springeropen.com/articles/10.1186/s40561-022-00192-z (дата звернення: 11.10.2023).

15.Стюарт Рассел. Сумісний з людиною. Штучний інтелект і проблема контролю, BookChef, 2020. 416с.

16. Alexeyev A. N. Multi–level test control of quality of knowledge by quantitative parameters. Computer modeling and new technologies. Riga: 2007. № 3. P. 43–45.

17. E. Walters. Using UML Activities to model Business Processes – Informatica, 2019, 236 с.

18.Explore the UML sequence diagram – IBM Developer: https:// https://developer.ibm.com/articles/the-sequence-diagram/ (дата звернення: 30.05.2023).

19.UML Revision Task Force. OMG Unified Modeling Language Specification, Version 1.4 (final draft). Берлін, 2001, 352 с.

20.Miro Samek. A crash course in UML state machines. California, 2015, 257p.

21. Principe J., Euliano N., Lefebvre W. Neural and Adaptive Systems. Fundamentals Through Simulations. John Wiley & Sons, New York. 2000, 672p.

22.Aurelien G. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems 3rd Edition. 2022, 850.

23.Foster Provost, Tom Fawcett, Data Science for Business: What You Need to Know about Data Mining and Data-Analytic Thinking, O'Reilly Media, 2013, 413p.

24.Cathy Chen, Niall Murphy, Kranti Parisa. Reliable Machine Learning. Applying SRE Principles to ML in Production, O'Reilly, 2022. 350p.

25.Scott V. Burger. Introduction to Machine Learning with R: Rigorous Mathematical Analysis, O'Reilly Media, 2018, 222p.

26.Christopher M. Bishop. Pattern Recognition and Machine Learning (Information Science and Statistics), Springer, 2006, 738p.

27.An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants: https://link.springer.com/article/10.1023/A:1007515423169 (дата звернення: 11.10.2023).

28.Anthony Sarkis. Training Data for Machine Learning. Human Supervision from Annotation to Data Science, O'Reilly, 2022. 250p.

29.Nikhil BudumaJoe PapaNithin Buduma. Fundamentals of Deep Learning. Designing Next-Generation Machine Intelligence Algorithms. 2nd Edition, O'Reilly, 2022. 388p.

30.Chip Huyen. Designing Machine Learning Systems: An Iterative Process for Production-Ready Applications, O'Reilly Media, 2022, 386p.

31.Baek C., Doleck T. Educational Data Mining: A Bibliometric Analysis of an Emerging Field. IEEE Access. 2022. 31289 – 31296pp.

32.Jonathon Manningm, Tim Nugent, Paris Buttfield-Addison, Mars Buttfield-Addison. Practical Simulations for Machine Learning. Using Synthetic Data for AI, O'Reilly, 2022. 332p.

33.Daniel Situnayake, Jenny Plunkett. AI at the Edge. Solving Real-World Problems with Embedded Machine Learning, O'Reilly Media, 2023, 512p.

34.Jeffrey Richter. CLR via C#. Washington, 2012, 896p.

35.Джон Скіт. C# для префесіоналів: тонкощі програмування, 3-є видання. Нью-Йорк, 2014, 608 с.

36.Troelsen A., Japikse P. Pro C# 7 With .NET and .NET Core. – 2017. 1372c.

37.Mark J. Price. C# 9 and .NET 5 - Modern Cross-Platform Development. Birmingham, 2023, 1328p

38.Visual Paradigm URL: https://www.visual-paradigm.com. (дата звернення: 30.05.2023).

39.Juval Lowy. Programming WCF Services: Design and Build Maintainable Service-Oriented Systems, Sebastopol, 2015, 1018p

40.Korotkevitch D. SQL Server Advanced Troubleshooting and Performance Tuning. Best Practices and Techniques, O'Reilly Media, 2022. 500p.

41.Kalen Delaney, Craig Freedman. Microsoft SQL Server 2012 Internals, Microsoft Press, 2013. 982p.

42.Alan Beaulieu. Learning SQL: Master SQL Fundamentals 3rd Edition, O'Reilly Media, 2020. 380p.

43.Curtis Tsang. Object-Oriented Technology from Diagram to Code with Visual Paradigm for UML, 1st edition. Лондон, 2005, 456 с.

44.Visual Studio: IDE and Code Editor for Software Developers and Teams: https://visualstudio.microsoft.com/ (дата звернення: 30.05.2023).

45.Joseph Albahari, Albahari. C# 9.0 Pocket Reference". Sebastopol, 2021, 264p.

46.Matthew McDonald. Pro WPF 4.5 in C# 2012: Windows Presentation Foundation in .NET 4.5, 4th edition. Лондон, 2013, 1024 с.

47.Майєрс Г. Мистецтво тестування програм. Чикаго, 2012, 272 с.

48.Jamie L Mitchell, Rex Black. Advanced Software Testing - Vol. 3, 2nd Edition. Guide to the ISTQB Advanced Certification as an Advanced Technical Test Analyst 2nd Edition, Rocky Nook, 2015. 480p.

49.Rex Black, Dorothy Graham, Erik van Veenendaal. Foundations of Software Testing ISTQB Certification, 4th edition, Cengage Learning EMEA, 2020. 288p.

50.Lee Copeland. A Practitioner's Guide to Software Test Design, Artech House, 2003. 300p.

51.Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. – Вінниця : ВНТУ, 2021. – 42 с. 35.

52.Кавецький В. В. Економічне обґрунтування інноваційних рішень: практикум / В. В. Кавецький, В. О. Козловський, І. В. Причепа – Вінниця : ВНТУ, 2016. – 113 с.

ДОДАТКИ

ДОДАТОК А

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ

д.т.н., проф. О. Н. Романюк

"19" 09 2023 р.

Технічне завдання

на магістерську кваліфікаційну роботу «Розробка методів і програмного засобу для системи адаптивного тестування знань» за спеціальністю

121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:

_____ д.т.н., проф. Ліщинська Л. Б.

"19" 09 2023 р.

Виконав:

_____студент гр.2ПІ-22м Ситніков Є. О.

"19" 09 2023 р.

Вінниця – 2023 року

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Розробка методів і програмного засобу для системи адаптивного тестування знань».

Галузь застосування – системи адаптивного тестування знань.

2. Підстава для розробки.

Підставою для виконання магістерської кваліфікаційної роботи (МКР) є індивідуальне завдання на МКР та наказ № 247 від « 18 » вересня 2023 р. ректора по ВНТУ про закріплення тем МКР.

3. Мета та призначення розробки.

Метою дослідження у роботі є підвищення ефективності тестування знань, зокрема в освітній системі за рахунок використання технологій машинного навчання.

Призначення роботи – розробка методів та програмного засобу для адаптивного тестування знань.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись МКР.

1. Scott A. Crossley, Danielle S. McNamara, Scott A. Crossley, Danielle S. McNamara. Adaptive Educational Technologies for Literacy Instruction, Routledge, 2016, 310p.

2. Drasgow F., Olson-Buchanan, J. B. Innovations in computerized assessment. Hillsdale, NJ: Erlbaum. 2006, 280p

3. Baek C., Doleck T. Educational Data Mining: A Bibliometric Analysis of an Emerging Field. IEEE Access. 2022. 31289 – 31296pp.

5. Технічні вимоги

Середовища розробки – Microsoft Visual Studio 2022, Weka, SQL Server Management Studio, мова розробки – C#, операційна система – Windows; методи

навчання моделі штучного інтелекту – класифікаційні; алгоритми навчання моделі – дерево рішень, найближчих сусідів, класифікатор Байєса.

6. Конструктивні вимоги.

Конструкція пристрою повинна відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

– пояснювальна записка до МКР;

– технічне завдання;

– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної галузі та вимог до створення системи	20.09 – 11.10	
2	Проектування програмного додатку	11.10 – 18.10	
3	Підготовка та навчання моделі ШІ	18.10 – 01.11	
4	Обґрунтування засобів для розробки додатку	01.11 – 03.11	
5	Розробка програмного засобу системи	03.11 – 20.11	
6	Економічна частина	20.11 – 24.11	
7	Оформлення матеріалів до захисту МКР	24.11 – 01.12	

10. Порядок контролю та прийняття.

Виконання етапів магістерської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи. Прийняття магістерської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

ДОДАТОК Б
ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: **Розробка методів і програмного засобу для системи адаптивного тестування знань.**

Тип роботи: кваліфікаційна робота

Підрозділ : кафедра програмного забезпечення, ФІТКІ, 2ПІ – 22м

Науковий керівник: д.т.н., проф. Ліщинська Л. Б.

Unicheck	
Оригінальність	**93.5%**
Схожість	6.5%

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Опис прийнятого рішення: **допустити до захисту**

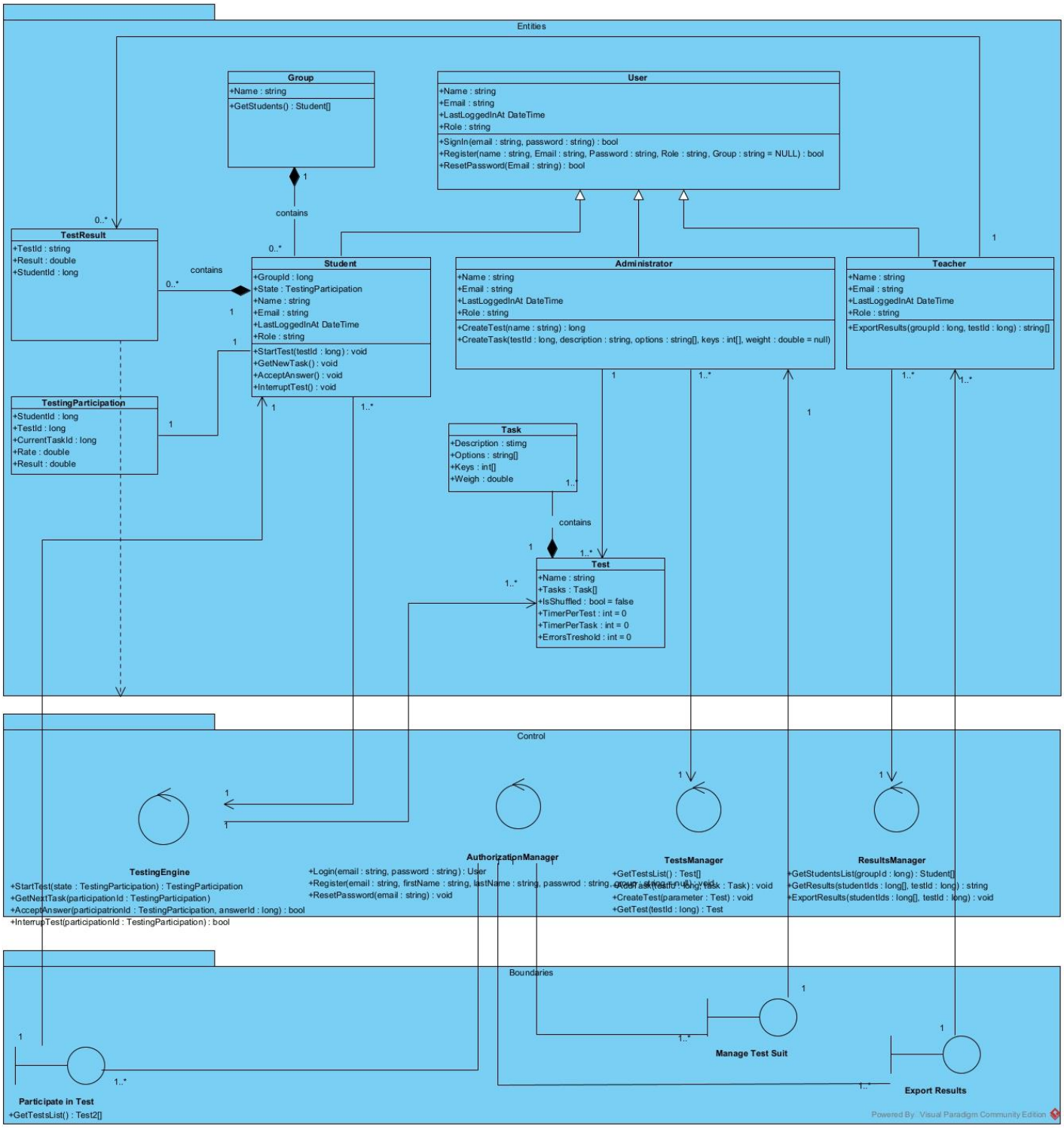
Ознайомлений з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Ситніков Є. О.

Керівник роботи _____ Ліщинська Л. Б.

ДОДАТОК В

Діаграма класів системи



ДОДАТОК Г

Лістинг коду програми

```
static class Program
   {
      /// <summary>
      /// The main entry point for the application.
      /// </summary>
      [STAThread]
      static void Main()
      {
         System.Windows.Forms.Application.EnableVisualStyles();
         System.Windows.Forms.Application.SetCompatibleTextRenderingDefault(false);
         StartForm start = new StartForm();
         System.Windows.Forms.Application.Run(start);
      }
   }

   [ServiceBehavior(ConcurrencyMode = ConcurrencyMode.Multiple, InstanceContextMode =
InstanceContextMode.PerSession, AutomaticSessionShutdown=false)]
        class ServiceController : ITestingServiceContract
        {
                TestingEngine _engine;
                User currentUser;
                AuthorizationManager _auth = AuthorizationManager.Instance;
                TestsManager _tests = TestsManager.Instance;
                UsersManager _users = UsersManager.Instance;
                SettingsManager _settings = SettingsManager.Instance;
                public ConnectionResponse Connect()
                {
                        OperationContext.Current.Channel.Closed += new
EventHandler(Channel_Closed);
                        OperationContext.Current.Channel.Faulted += new
EventHandler(Channel_Closed);
                        var callbackChannel =
OperationContext.Current.GetCallbackChannel<ITestingServiceCallback>();

                        _settings.OnSettingsChanged += (sender, settings) =>
callbackChannel.OnChangeSettings(settings); //subscribe UI on config changes
                        _tests.OnTestsChanged += (sender, list) =>
callbackChannel.OnTestsChanged(list); //subscribe UI on tests changes
                        _users.OnGroupsChanged += (sender, list) =>
callbackChannel.OnGroupsChanged(list); //subscribe UI on groups changes

                        var groups = _users.GetGroupsList();
                        var tests = _tests.GetTestsList();
                        return new ConnectionResponse
                        {
                                Groups = groups,
                                Tests = tests
                        };
                }
```

```
        public void Channel_Closed(object sender, EventArgs e)
        {
                _auth.Logout(currentUser?.Name);
                _engine.InterrupTest();
                Disconnect();
        }
        public Settings Login(string name, string password)
        {
                var result = _auth.Login(name, password);
                if (result == null) return null;
                currentUser = result;
                _engine = new TestingEngine(currentUser);
                return _settings.CurrentSettings;
        }

        public void Logout()
        {
                _auth.Logout(currentUser?.Name);
                currentUser = null;
        }
        public bool Registration(string login, string password, string group)
        {
                return _auth.Registration(login, password, group);
        }
        public void ConfirmAnswer(string answer)
        {
                _engine.AcceptAnswer(answer);
        }
        public void StartTesting(string name)
        {
                _engine.StartTest(name);
        }
        public string[] GetStudentsList(string groupName)
        {
return _users.GetStudentsList(groupName);
        }
        public Task GetTask()
        {
                return _engine.GetNextTask();
        }
        public TestingParticipation GetState()
        {
                return _engine.GetState();
        }
        public void FastFinish()
        {
                _engine.InterrupTest();
        }
        public void Disconnect()
        {
        }
}
```

```
  public class Student : User
    {
       public string Group;

       public TestingParticipation State { get; set; }
    }

    [DataContract]
    public class TestingParticipation
    {
       [DataMember]
       public string Name { get; set; }//+
       [DataMember]
       public int QuestionsCount { get; set; }//+
       [DataMember]
       public int QuestionNumber { get; set; }//+
       [DataMember]
       public double Difficult { get; set; }//+
       [DataMember]
       public double Rate { get; set; }//+
       [DataMember]
       public double MaxResult { get; set; }//+
       [DataMember]
       public double Result { get; set; }//+
       [DataMember]
       public double ResultPersent { get; set; }//+
       [DataMember]
       public int? AllowedMistakes { get; set; }//+
       [DataMember]
       public long StudentId { get; set; }
       [DataMember]
       public long TestId { get; set; }
       [DataMember]
       public long CurrentTaskId { get; set; }
    }

public class UsersManager
    {
       public event EventHandler<string[]> OnGroupsChanged;
       SQLDataManager _dbManager = SQLDataManager.GetInstatnce;

       private UsersManager()
       { }

       public static UsersManager Instance { get; } = new UsersManager();

       public string[] GetStudentsList(string groupName)
       {
          return _dbManager.GetStudentsFromGroup(groupName);
       }

       public string[] GetAdminsList()
```

```
        {
            return _dbManager.GetAdminsList();
        }

        public string[] GetGroupsList()
        {
            return _dbManager.GetGroupsList();
        }

        public bool AddGroup(string name)
        {
            if (!_dbManager.AddGroup(name)) return false;
            var groups = GetGroupsList();
            OnGroupsChanged?.Invoke(this, groups);
            return true;
        }

        public void DeleteGroup(string name)
        {
            _dbManager.DeleteGroup(name);
        }
    }

    public class TestsManager
    {
        public event EventHandler<string[]> OnTestsChanged;
        private TestsManager() { }
        public static TestsManager Instance { get; } = new TestsManager();

        SQLDataManager _dbManager = SQLDataManager.GetInstatnce;

        public string[] GetTestsList()
        {
            return _dbManager.GetTestsList();
        }

        public void AddTask(string testName, string question, string[] answers, string answer, string
picturePath, double weight, int type)
        {
            _dbManager.AddQuestionToTest(testName, question, answers, answer, picturePath, weight,
type);
        }

        public bool CreateTest(string name)
        {
            if (_dbManager.AddTest(name))
            {
                OnTestsChanged?.Invoke(this, _dbManager.GetTestsList());
                return true;
            }
            return false;
        }
```

```
        public Task[] GetTest(string name)
        {
            return _dbManager.GetTest(name).ToArray();
        }

        public (int questionsCount, double scores) GetTestGeneralInfo(string name)
        {
            var scores  = _dbManager.GetPointsForTest(name);
            var questionsCount = _dbManager.GetQuestionsCount(name);
            return (questionsCount, scores);
        }
    }


public class TestingEngine
    {
        SQLDataManager _dbManager = SQLDataManager.GetInstatnce;
        List<Task> _currentTest = null;
        string _currentTestName = null;
        Task _currentQuestion = null;
        TestingParticipation _currentState;
        ModelWrapper model = new ModelWrapper();
        public TestingEngine(User user)
        {
            _currentState = new TestingParticipation
            {
                Name = user.Name
            };
        }
        public void Shuffling()
        {
            List<Task> shuffledList = new List<Task>();
            Random rnd = new Random(Environment.TickCount);
            for (int i = 0; _currentTest.Count > 0; i++)
            {
                int n = rnd.Next(0, _currentTest.Count);
                shuffledList.Add(_currentTest[n]);
                _currentTest.RemoveAt(n);
            }
            _currentTest = shuffledList;
        }
        public void Correction(bool result)
        {
            double difficult = _currentQuestion.Weight;
            if (result) difficult /= 1.05f;
            else difficult *= 1.05f;
            difficult = Math.Round(difficult, 5);
            if (difficult < 1) difficult = 1;
            if (difficult > 5) difficult = 5;
            _dbManager.UpdateWeight(_currentTestName, _currentQuestion.Question, difficult);
        }
        public void StartTest(string testName)
        {
```

```
            _currentState.Results = new List<double>();
            _currentState.MaxResult = 0;
            _currentState.Result = 0;
            _currentState.ResultPersent = 0;
            _currentTest = _dbManager.GetTest(testName);
            _currentTestName = testName;
            _currentState.AllowedMistakes =
SettingsManager.Instance.CurrentSettings.AllowedMistakes;
            _currentState.QuestionNumber = 0;
            if (SettingsManager.Instance.CurrentSettings.TestShuffling) Shuffling();
            if (SettingsManager.Instance.CurrentSettings.SelectFragment != null)
            {
               if (SettingsManager.Instance.CurrentSettings.SelectFragment <= _currentTest.Count())
_currentState.QuestionsCount = (int)SettingsManager.Instance.CurrentSettings.SelectFragment;
               else _currentState.QuestionsCount = _currentTest.Count();
            }
            else _currentState.QuestionsCount = _currentTest.Count();
            AdminLogger.Instance.Log(DateTime.Now.ToString() + ": " + _currentState.Name + "
розпочав(ла) тест " + testName);
         }

         public Task GetNextTask()
         {
            SelectQuestion();
            Task task = null;
            if (_currentQuestion != null)
            {
               task = (Task)_currentQuestion.Clone();
               task.Answer = null;
               var compression = SettingsManager.Instance.CurrentSettings.ImagesCompression;
               if (task.Image != null && compression != 0) task.Image = Compress(task.Image, 100 -
compression);
            }
            return task;
         }
         public TestingParticipation GetState()
         {
            return _currentState;
         }

         public void SelectQuestion()
         {
            if (_currentState.QuestionNumber != _currentState.QuestionsCount)
            {
               if (SettingsManager.Instance.CurrentSettings.Adapting == true)
               {
                  double currentDelta = 0;
                  double delta = Math.Abs(_currentTest[0].Weight - _currentState.Rate);
                  _currentQuestion = _currentTest[0];
                  foreach (Task task in _currentTest)
                  {
                     currentDelta = Math.Abs(task.Weight - _currentState.Rate);
                     if (delta > currentDelta)
```

```
                {
                    delta = currentDelta; _currentQuestion = task;
                }
            }
            _currentState.Difficult = _currentQuestion.Weight;
        }
        else _currentQuestion = _currentTest[0];
        _currentState.QuestionNumber++;
        _currentState.Difficult = _currentQuestion.Weight;
    }
    else
    {
        _currentQuestion = null;
        SaveResults(_currentState.Name, _currentState.Rate, _currentState.Result,
_currentState.ResultPersent, _currentTestName);
        AdminLogger.Instance.Log(DateTime.Now.ToString() + ": " + _currentState.Name + "
завершив(ла) тест " + _currentTestName);
    }
}
public void AcceptAnswer(string answer)
{
    double result = 0;
    if (answer == _currentQuestion.Answer)
    {
        result = _currentQuestion.Weight;
        if (SettingsManager.Instance.CurrentSettings.Adapting == true) _currentState.Result +=
_currentQuestion.Weight;
        else _currentState.Result++;
        if (SettingsManager.Instance.CurrentSettings.AutoCorrection) Correction(true);

        _currentState.Rate *= 1.2;
        if (_currentState.Rate > 5) _currentState.Rate = 5;
    }
    else
    {
        if (SettingsManager.Instance.CurrentSettings.AutoCorrection) Correction(false);
        _currentState.Rate /= 1.2;
        if (_currentState.Rate < 1) _currentState.Rate = 1;
        if (_currentState.AllowedMistakes >= 0 && _currentState.AllowedMistakes != null)
        {
            _currentState.AllowedMistakes--;
        }
    }
    _currentTest.Remove(_currentQuestion);
    if (SettingsManager.Instance.CurrentSettings.Adapting == true)
    {
        _currentState.Results.Add(result);
        _currentState.MaxResult += _currentQuestion.Weight;
        var scoring = model.ScoreModel(_currentState.Results.ToArray());
        if(scoring.distribution >= SettingsManager.Instance.CurrentSettings.ScoringTreshold)
        {
            _currentState.ResultPersent = scoring.result;
            InterrupTest();
```

```
            }
        }
        else _currentState.MaxResult++;
        _currentState.ResultPersent = Math.Round(_currentState.Result / _currentState.MaxResult
* 100, 2);
    }

    public void InterrupTest()
    {
        //TestTask task = new TestTask();
        while (_currentQuestion != null)
        {
            SelectQuestion();
            if (_currentQuestion != null) AcceptAnswer(null);
        }
    }

    public void SaveResults(string name, double rate, double res, double pers, string testName)
    {
        _dbManager.SaveResults(name, rate, res, pers, testName);
        AdminLogger.Instance.UpdateResults();
    }

    private MemoryStream Compress(MemoryStream stream, long value)
    {
        Image bmp = Image.FromStream(stream);
        ImageCodecInfo jgpEncoder = GetEncoder(ImageFormat.Jpeg);
        System.Drawing.Imaging.Encoder myEncoder =
System.Drawing.Imaging.Encoder.Quality;
        EncoderParameters myEncoderParameters = new EncoderParameters(1);
        EncoderParameter myEncoderParameter = new EncoderParameter(myEncoder, value);
        myEncoderParameters.Param[0] = myEncoderParameter;
        MemoryStream resultStream = new MemoryStream();
        bmp.Save(resultStream, jgpEncoder, myEncoderParameters);
        //TypeConverter bmpConverter = TypeDescriptor.GetConverter(bmp.GetType());
        //byte[] bmpData = (byte[])bmpConverter.ConvertTo(bmp, typeof(byte[]));
        //pictureBoxImage.Image = bmp;
        return resultStream;
    }

    public static ImageCodecInfo GetEncoder(ImageFormat format)
    {
        var codecs = ImageCodecInfo.GetImageDecoders();
        return codecs.FirstOrDefault(codec => codec.FormatID == format.Guid);
    }
  }
}

  public class SettingsManager
  {
    public event EventHandler<Settings> OnSettingsChanged;

    private SettingsManager()
```

```
        {
            _settings = XmlManager.ReadFromFile();
        }

        public static SettingsManager Instance { get; } = new SettingsManager();

        private Settings _settings;

        public void SetSettings(Settings newSettings)
        {
            _settings = newSettings;
            _settings.SaveToFile();
            OnSettingsChanged?.Invoke(this, newSettings);
        }

        public Settings CurrentSettings => _settings;
    }

public class ResultsManager
    {
        private ResultsManager()
        { }
        public static ResultsManager Instance { get; } = new ResultsManager();

        SQLDataManager _dbManager = SQLDataManager.GetInstatnce;
        public Student[] GetStudentsList(long groupId)
        {
            throw new NotImplementedException("Not implemented");
        }
        public DataTable GetResults(string groupName)
        {
            return _dbManager.GetResults(groupName);
        }
        public void ExportResults(DataTable source, FileStream fs)
        {
            StringBuilder sb = new StringBuilder();
            sb.AppendLine("Ім'я, Тест, Результат, Успішність, Дата");
            foreach (DataRow row in source.Rows)
            {
                IEnumerable<string> fields = row.ItemArray.Select(field => field.ToString());
                sb.AppendLine(string.Join(",", fields));
            }
            var data = Encoding.UTF8.GetBytes(sb.ToString());
            byte[] content = Encoding.UTF8.GetPreamble().Concat(data).ToArray();
            fs.Write(content, 0, content.Length);
            fs.Close();
        }
    }

internal class ModelWrapper
    {
        private Classifier _model;
        public ModelWrapper()
```

```
        {
            _model = (Classifier)weka.core.SerializationHelper.read("testingModel.model");
        }

        private Instances PrepareInput(double[] answers)
        {
            Instances inputs = new Instances(new java.io.FileReader("modelInput.arff"));
            inputs.setClassIndex(inputs.numAttributes() - 1);
            return inputs;
        }

        public (double result, double distribution) ScoreModel(double[] answers)
        {
            Random a = new Random();
            try
            {
                var results = new List<(double result, double prob)>();
                var input = PrepareInput(answers).firstInstance();
                var distribution = _model.distributionForInstance(input);
                var result = _model.classifyInstance(input);
                var prob = distribution.Max(x => x);

                return (result, prob);
            }
            catch (Exception e)
            {
                return (0, 0);
            }
        }
    }

public class SQLDataManager
    {
        private static SQLDataManager _manager = new SQLDataManager();
        public static SQLDataManager GetInstatnce
        { get { return _manager; } }

        string _ConnectionString =
ConfigurationManager.ConnectionStrings["conString"].ToString();
        public SqlConnection returnConnection()
        {
            SqlConnection myConnection = new SqlConnection(_ConnectionString);
            return myConnection;
        }
        public bool Login(string name, string password)
        {
            using (SqlConnection myConnection = returnConnection())
            {
                myConnection.Open();
                SqlCommand cmd = new SqlCommand("Login", myConnection);
                cmd.CommandType = CommandType.StoredProcedure;
                cmd.Parameters.AddWithValue("@Name", name);
                cmd.Parameters.AddWithValue("@Password", password);
```

```
            if (cmd.ExecuteNonQuery() == 1) return true;
            else return false;
        }
    }

    public User AdminLogin(string name, string password)
    {
        using (SqlConnection myConnection = returnConnection())
        {
            myConnection.Open();
            SqlCommand cmd = new SqlCommand("Select top 1 * from Admins where name =
@Name and password = @Password", myConnection);
            cmd.CommandType = CommandType.Text;
            cmd.Parameters.AddWithValue("@Name", name);
            cmd.Parameters.AddWithValue("@Password", password);
            SqlDataReader reader = cmd.ExecuteReader();
            if (reader.Read())
            {
                return new User
                {
                    Name = reader["name"].ToString(),
                    Role = (Roles)Enum.Parse(typeof(Roles), reader["role"].ToString()),
                    LastLoggedInAt = DateTime.Now
                };
            }
            return null;
        }
    }

    public string[] GetAdminsList()
    {
        List<string> admins = new List<string>();
        using (SqlConnection myConnection = returnConnection())
        {
            myConnection.Open();
            SqlCommand cmd = new SqlCommand("Select name from Admins", myConnection);
            cmd.CommandType = CommandType.Text;
            SqlDataReader reader = cmd.ExecuteReader();
            while (reader.Read())
            {
                admins.Add((string)reader["name"]);
            }
        }
        return admins.ToArray();
    }

    public void Logout(string name)
    {
        using (SqlConnection myConnection = returnConnection())
        {
            myConnection.Open();
            SqlCommand cmd = new SqlCommand("Logout", myConnection);
            cmd.CommandType = CommandType.StoredProcedure;
```

```
            cmd.Parameters.AddWithValue("@Name", name);
            cmd.ExecuteNonQuery();
        }
    }
    public bool Registration(string name, string password, string group)
    {
        using (SqlConnection myConnection = returnConnection())
        {
            myConnection.Open();
            SqlCommand cmd = new SqlCommand("Registration", myConnection);
            cmd.CommandType = CommandType.StoredProcedure;
            cmd.Parameters.AddWithValue("@Name", name);
            cmd.Parameters.AddWithValue("@Password", password);
            cmd.Parameters.AddWithValue("@GroupName", group);
            if (cmd.ExecuteNonQuery() == 1)
            {
                return true;
            }
            else return false;
        }
    }

    public bool AdminRegistration(string name, string password, Roles role)
    {
        using (SqlConnection myConnection = returnConnection())
        {
            myConnection.Open();
            SqlCommand cmd = new SqlCommand("Insert into Admins (name, password, role)
values (@Name, @Password, @Role)", myConnection);
            cmd.CommandType = CommandType.Text;
            cmd.Parameters.AddWithValue("@Name", name);
            cmd.Parameters.AddWithValue("@Password", password);
            cmd.Parameters.AddWithValue("@Role", role);
            if (cmd.ExecuteNonQuery() == 1)
            {
                return true;
            }
            else return false;
        }
    }
    public bool AddGroup(string name)
    {
        using (SqlConnection myConnection = returnConnection())
        {
            myConnection.Open();
            SqlCommand cmd = new SqlCommand("AddGroup", myConnection);
            cmd.CommandType = CommandType.StoredProcedure;
            cmd.Parameters.AddWithValue("@Name", name);
            if (cmd.ExecuteNonQuery() == 1)
            {
                return true;
            }
            else return false;
```

```
        }
    }
    public void DeleteGroup(string name)
    {
        using (SqlConnection myConnection = returnConnection())
        {
            myConnection.Open();
            SqlCommand cmd = new SqlCommand("DelGroup", myConnection);
            cmd.CommandType = CommandType.StoredProcedure;
            cmd.Parameters.AddWithValue("@Name", name);
            cmd.ExecuteNonQuery();
        }
    }
    public bool AddTest(string name)
    {
        using (SqlConnection myConnection = returnConnection())
        {
            myConnection.Open();
            SqlCommand cmd = new SqlCommand("CheckTestExisting", myConnection);
            cmd.CommandType = CommandType.StoredProcedure;
            cmd.Parameters.AddWithValue("@Name", name);
            SqlDataReader reader = cmd.ExecuteReader();
            reader.Read();
            int countOfSuchTables = (int)reader[0];
            reader.Close();
            if (countOfSuchTables == 0)
            {

                cmd = new SqlCommand("AddTest", myConnection);
                cmd.CommandType = CommandType.StoredProcedure;
                cmd.Parameters.AddWithValue("@Name", name);
                cmd.ExecuteNonQuery();
                return true;
            }
            else return false;
        }
    }
    public void AddQuestionToTest(string testName, string question, string[] answers, string
answer, string picturePath, double weight, int type)
    {
        string parsedWeight = weight.ToString().Replace(",", ".");
        using (SqlConnection myConnection = returnConnection())
        {
            myConnection.Open();
            SqlCommand cmd = new SqlCommand("insert into " + testName + "
(question,answer1,answer2,answer3,answer4,answer,picturePath,weight,type) values ('" + question
+ "','" + answers[0] + "','" + answers[1] + "','" + answers[2] + "','" + answers[3] + "','" + answer +
"','" + picturePath + "','" + weight + "','" + type + "')", myConnection);
            cmd.ExecuteNonQuery();
        }
    }
    public int GetQuestionsCount(string testName)
    {
```

```
    using (SqlConnection myConnection = returnConnection())
    {
        myConnection.Open();
        SqlCommand cmd = new SqlCommand("GetQuestionsCount", myConnection);
        cmd.CommandType = CommandType.StoredProcedure;
        cmd.Parameters.AddWithValue("@Name", testName);
        SqlDataReader reader = cmd.ExecuteReader();
        reader.Read();
        return (int)reader[0];
    }
}
public double GetPointsForTest(string testName)
{
    using (SqlConnection myConnection = returnConnection())
    {
        double pointsSum = 0;
        myConnection.Open();
        SqlCommand cmd = new SqlCommand("GetPointsFromTest", myConnection);
        cmd.CommandType = CommandType.StoredProcedure;
        cmd.Parameters.AddWithValue("@Name", testName);
        SqlDataReader reader = cmd.ExecuteReader();
        while (reader.Read())
        {
            pointsSum += (double)reader["weight"];
        }
        return pointsSum;
    }
}
public string[] GetStudentsList()
{
    using (SqlConnection myConnection = returnConnection())
    {
        List<string> result = new List<string>();
        myConnection.Open();
        SqlCommand cmd = new SqlCommand("GetStudentsList", myConnection);
        cmd.CommandType = CommandType.StoredProcedure;
        SqlDataReader reader = cmd.ExecuteReader();
        while (reader.Read())
        {
            result.Add((string)reader["name"]);
        }

        return result.ToArray();
    }
}
public string[] GetStudentsFromGroup(string groupName)
{
    using (SqlConnection myConnection = returnConnection())
    {
        List<string> result = new List<string>();
        myConnection.Open();
        SqlCommand cmd = new SqlCommand("GetStudentsFromGroup", myConnection);
        cmd.CommandType = CommandType.StoredProcedure;
```

```
        cmd.Parameters.AddWithValue("@GroupName", groupName);
        SqlDataReader reader = cmd.ExecuteReader();
        while (reader.Read())
        {
            result.Add((string)reader["name"]);
        }
        return result.ToArray();
    }
}
public string[] GetGroupsList()
{
    using (SqlConnection myConnection = returnConnection())
    {
        List<string> result = new List<string>();
        myConnection.Open();
        SqlCommand cmd = new SqlCommand("GetGroupsList", myConnection);
        cmd.CommandType = CommandType.StoredProcedure;
        SqlDataReader reader = cmd.ExecuteReader();
        while (reader.Read())
        {
            result.Add((string)reader["name"]);
        }
        return result.ToArray();
    }
}
public string[] GetTestsList()
{
    try
    {
        using (SqlConnection myConnection = returnConnection())
        {
            List<string> result = new List<string>();
            myConnection.Open();
            SqlCommand cmd = new SqlCommand("GetTestsList", myConnection);
            cmd.CommandType = CommandType.StoredProcedure;
            SqlDataReader reader = cmd.ExecuteReader();
            while (reader.Read())
            {
                result.Add((string)reader["name"]);
            }
            return result.ToArray();
        }
    }
    catch { return null; }
}
public List<Task> GetTest(string name)
{
    using (SqlConnection myConnection = returnConnection())
    {
        List<Task> result = new List<Task>();
        string[] answersArray = new string[4];
        myConnection.Open();
        SqlCommand cmd = new SqlCommand("GetTest", myConnection);
```

```
            cmd.CommandType = CommandType.StoredProcedure;
            cmd.Parameters.AddWithValue("@Name", name);
            SqlDataReader reader = cmd.ExecuteReader();
            while (reader.Read())
            {
               Task task = new Task();
               task.answers[0] = (string)reader["answer1"];
               task.answers[1] = (string)reader["answer2"];
               task.answers[2] = (string)reader["answer3"];
               task.answers[3] = (string)reader["answer4"];
               task.Question = (string)reader["question"];
               task.Answer = (string)reader["answer"];
               task.Image = ReturnFileStream((string)reader["picturePath"]);
               task.Type = (QuestionType)reader["type"];
               task.Weight = (double)reader["weight"];
               result.Add(task);
            }
            return result;
         }
      }
      private MemoryStream ReturnFileStream(string filePath)
      {
         Image bmp;
         if (File.Exists(filePath))
         {
            bmp = Image.FromFile(filePath);
         }
         else return null;
         TypeConverter bmpConverter = TypeDescriptor.GetConverter(bmp.GetType());
         byte[] bmpData = (byte[])bmpConverter.ConvertTo(bmp, typeof(byte[]));
         MemoryStream stream = new MemoryStream(bmpData);
         return stream;
      }
      public double GetStudentRate(string name)
      {
         using (SqlConnection myConnection = returnConnection())
         {
            myConnection.Open();
            SqlCommand cmd = new SqlCommand("GetStudentRate", myConnection);
            cmd.CommandType = CommandType.StoredProcedure;
            cmd.Parameters.AddWithValue("@Name", name);
            SqlDataReader reader = cmd.ExecuteReader();
            reader.Read();
            return (double)reader["rate"];
         }
      }
      public void SaveResults(string name, double rate, double result, double resultPersent, string
testName)
      {
         using (SqlConnection myConnection = returnConnection())
         {
            myConnection.Open();
            SqlCommand cmd = new SqlCommand("SaveResults", myConnection);
```

```
            cmd.CommandType = CommandType.StoredProcedure;
            cmd.Parameters.AddWithValue("@Name", name);
            cmd.Parameters.AddWithValue("@TestName", testName);
            cmd.Parameters.AddWithValue("@Result", result);
            cmd.Parameters.AddWithValue("@ResultPersent", resultPersent);
            cmd.ExecuteNonQuery();

            cmd = new SqlCommand("UpdateRate", myConnection);
            cmd.CommandType = CommandType.StoredProcedure;
            cmd.Parameters.AddWithValue("@Name", name);
            cmd.Parameters.AddWithValue("@Rate", rate);
            cmd.ExecuteNonQuery();
        }
    }
    public void UpdateWeight(string name, string question, double weight)
    {
        string str = weight.ToString();
        str = str.Replace(",", ".");
        using (SqlConnection myConnection = returnConnection())
        {
            myConnection.Open();
            SqlCommand cmd = new SqlCommand("UPDATE " + name + " SET weight =" + str + "
WHERE question ='" + question + "'", myConnection);
            cmd.ExecuteNonQuery();
        }
    }
    public void GlobalLogout()
    {
        using (SqlConnection myConnection = returnConnection())
        {
            myConnection.Open();
            SqlCommand cmd = new SqlCommand("LogoutAll", myConnection);
            cmd.CommandType = CommandType.StoredProcedure;
            cmd.ExecuteNonQuery();
        }
    }
    public DataTable GetResults(string groupName)
    {
        using (SqlConnection myConnection = returnConnection())
        {
            myConnection.Open();
            SqlCommand cmd = new SqlCommand("GetResults", myConnection);
            cmd.CommandType = CommandType.StoredProcedure;
            cmd.Parameters.AddWithValue("@GroupName", groupName);
            SqlDataAdapter myAdapter = new SqlDataAdapter(cmd);
            DataTable table = new DataTable();
            myAdapter.Fill(table);
            return table;
        }
    }
}
```

ДОДАТОК Д

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА МЕТОДІВ І ПРОГРАМНОГО ЗАСОБУ ДЛЯ СИСТЕМИ АДАПТИВНОГО ТЕСТУВАННЯ ЗНАНЬ

РОЗРОБКА МЕТОДІВ І ПРОГРАМНОГО ЗАСОБУ ДЛЯ СИСТЕМИ АДАПТИВНОГО ТЕСТУВАННЯ ЗНАНЬ

Виконав:
магістрант Ситніков Є. О.
Керівник:
д.т.н., проф. Ліщинська Л. Б.

Рисунок Д.1 – Слайд презентації №1

Мета дослідження

- **Мета і завдання дослідження.** Метою дослідження у роботі є підвищення ефективності тестування знань, зокрема в освітній системі за рахунок використання технологій машинного навчання.
- Для досягнення поставленої мети необхідно виконати наступні завдання:
  - проаналізувати предметну галузь та існуючі рішення в галузі адаптивного тестування знань;
  - розробити метод для визначення моменту закінчення процесу тестування та оцінки знань користувача;
  - покращити метод адаптивності системи тестування для підлаштування завдань до рівня знань користувача;
  - підготувати та провести навчання моделі штучного інтелекту для застосування нового методу;
  - спроектувати та розробити програмне забезпечення для адаптивного тестування знань з використанням моделі штучного інтелекту ;
  - Провести тестування розробленої системи.
- **Об'єкт дослідження** – процеси адаптивного тестування знань.
- **Предмет дослідження** – методи та засоби адаптивного тестування знань.

Рисунок Д.2 – Слайд презентації №2

Класичний алгоритм адаптивної системи тестування знань

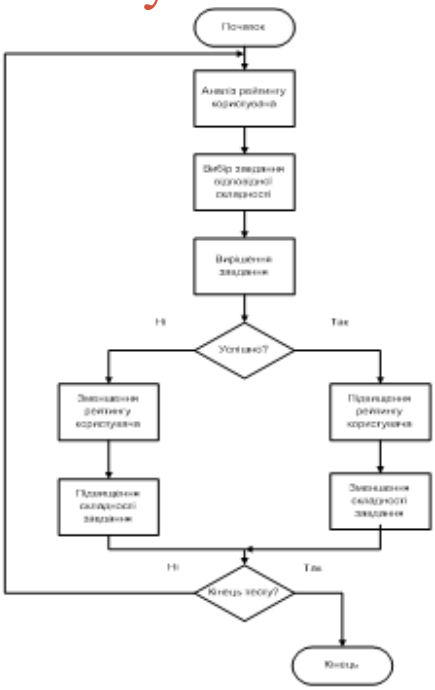


Рисунок Д.3 – Слайд презентації №3

Класичний графік підбору складності завдань

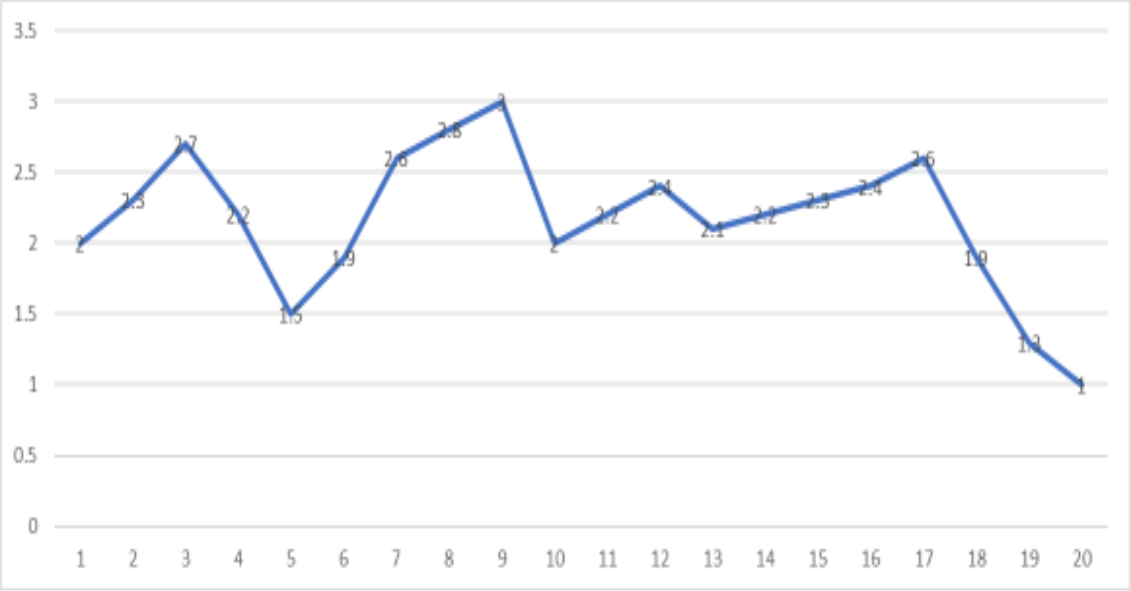


Рисунок Д.4 – Слайд презентації №4

Рисунок Д.5 – Слайд презентації №5

Рисунок Д.6 – Слайд презентації №6

Покращення методу підбору складності завдань

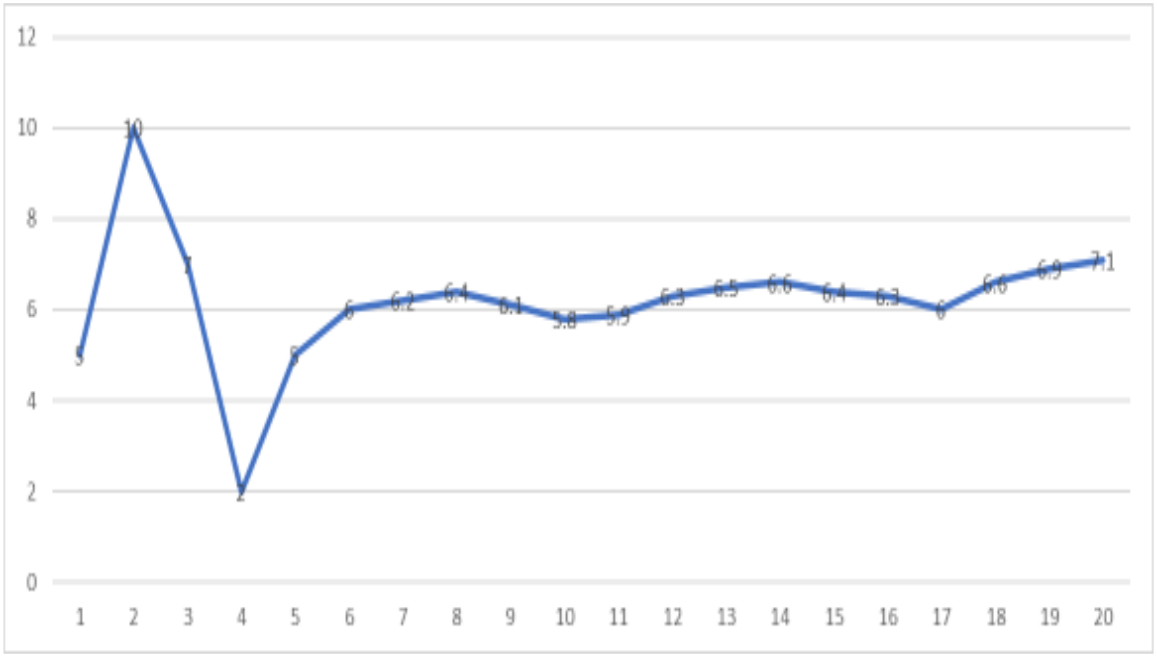


Рисунок Д.7 – Слайд презентації №7

Запропонований метод калібрування складності завдань

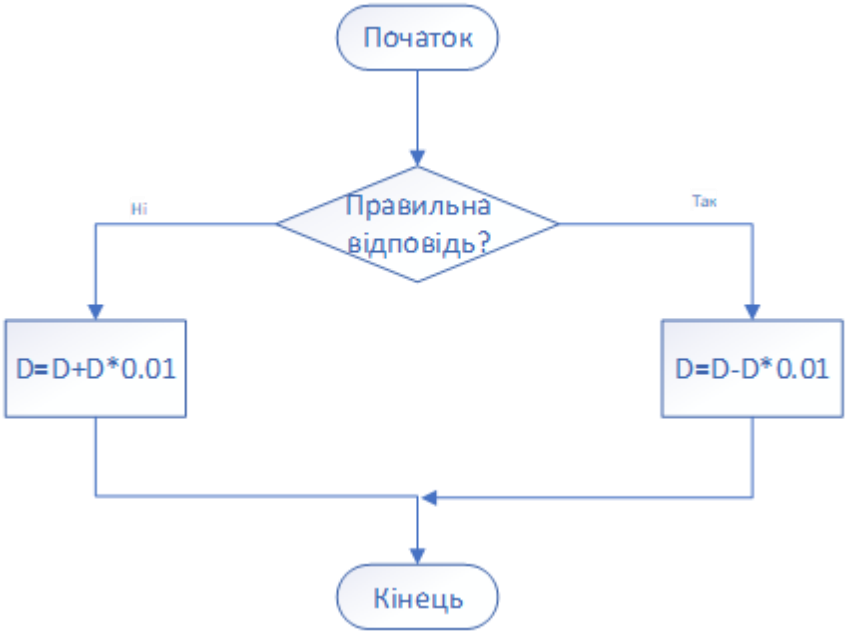
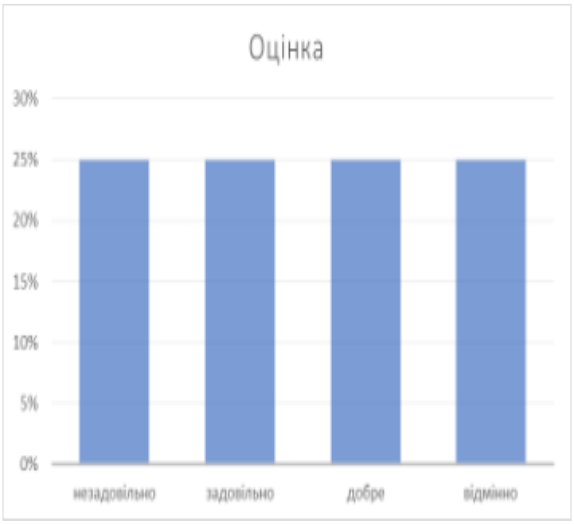


Рисунок Д.8 – Слайд презентації №8

Запропонований метод оцінки користувача та завершення тестування

Початок тестування:



Після проходженняN завдання:

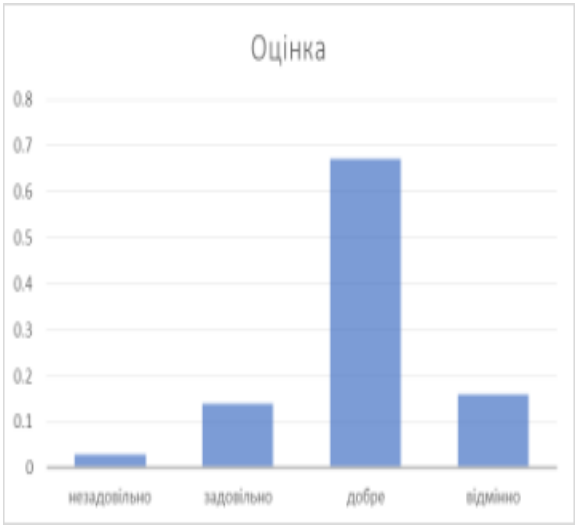
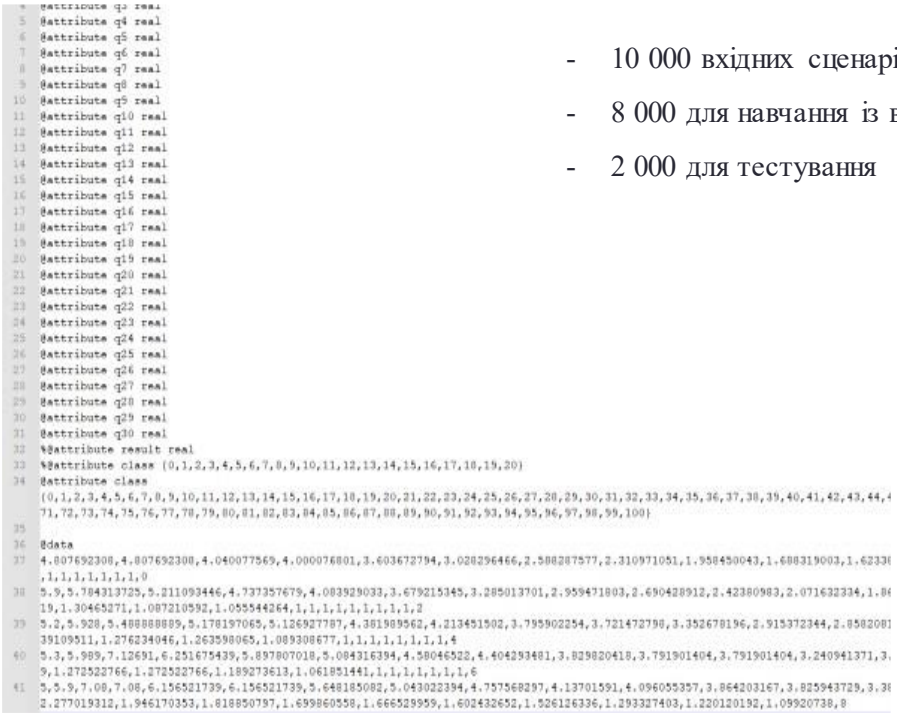


Рисунок Д.9 – Слайд презентації №9

Фрагмент навчального набору



- 10 000 вхідних сценаріїв для навчання
- 8 000 для навчання із вчителем
- 2 000 для тестування

Рисунок Д.10 – Слайд презентації №10

Фрагмент згенерованого дерева рішень

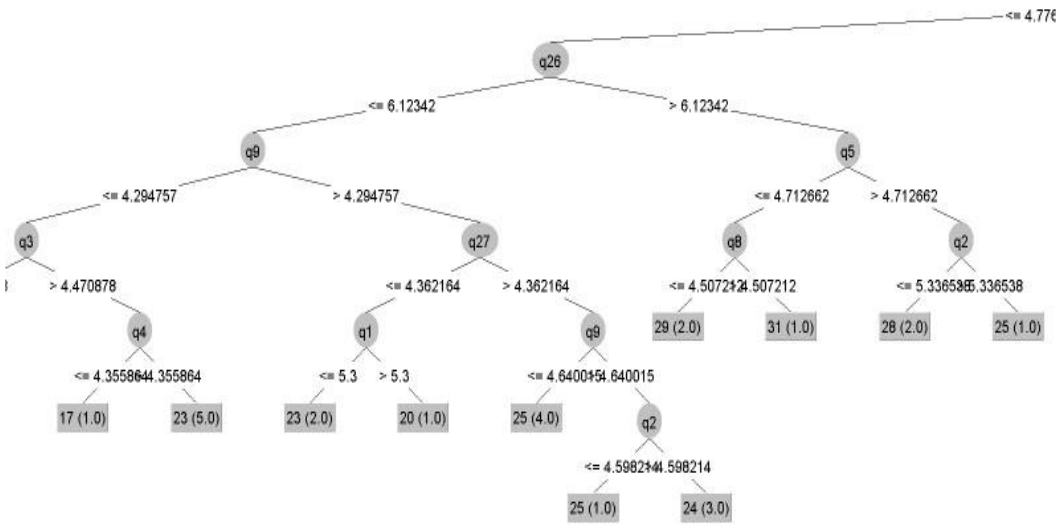


Рисунок Д.11 – Слайд презентації №11

Розподіл помилок класифікації

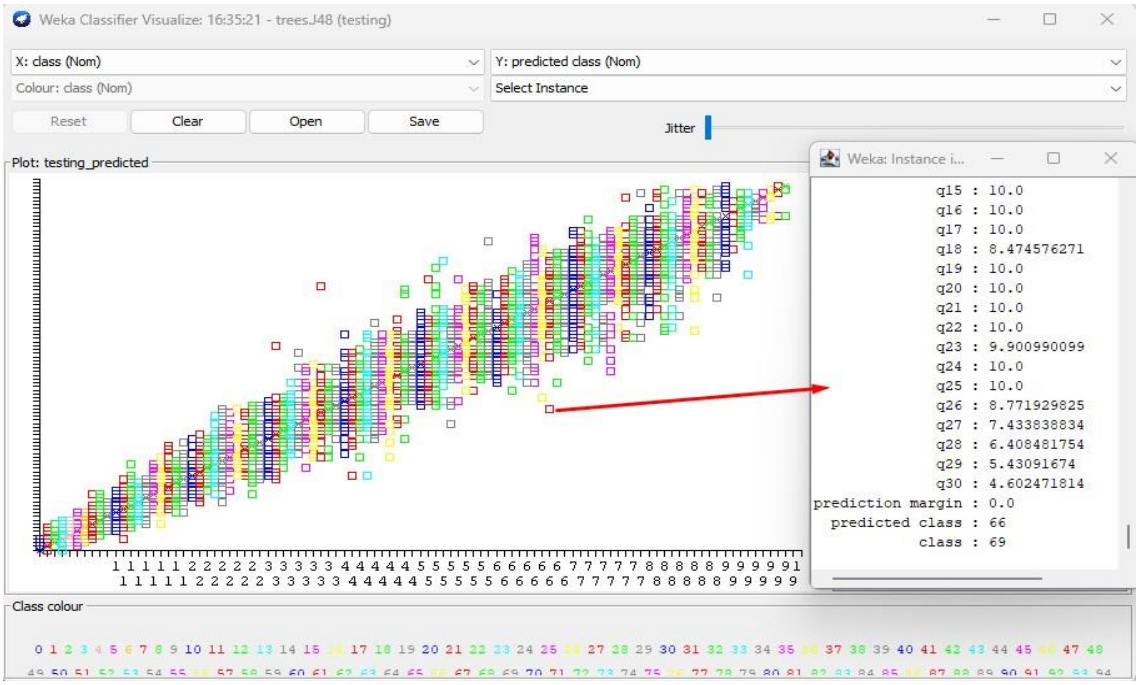


Рисунок Д.12 – Слайд презентації №12

Результати тестування розробленого методу оцінки користувачів

	A	B	C	D	E	F
1	Результат		Результат	Ймовірність	Кількість завдань	Похибка
2	39		35	76%	22	-10%
3	32		32	75%	22	0%
4	19		19	77%	23	0%
5	13		11	87%	26	-14%
6	44		46	84%	25	5%
7	15		16	84%	25	11%
8	18		18	81%	24	5%
9	6		6	73%	21	0%
10	28		28	80%	24	0%
11	21		19	84%	25	-5%
12	46		36	75%	22	-20%
13	42		42	93%	27	0%
14	27		27	83%	24	0%
15	11		11	77%	23	0%
16	54		55	88%	26	2%
17	18		15	70%	21	-13%
18	15		15	73%	21	0%
19	21		24	73%	21	18%
20	40		40	77%	23	0%
21	50		57	78%	23	14%
22	25		25	86%	25	1%
23	30		30	75%	22	3%
24	24		24	82%	24	0%
25	39		39	88%	26	0%
26	32		29	88%	26	-9%
27	48		45	79%	23	-5%
28	24		24	83%	24	0%
29	8		8	86%	25	6%
30	38		34	80%	24	-8%
31	36		42	87%	26	19%
32	15		15	93%	27	3%

- середнє відхилення результату склало 6%. Похибка коливалась від 0% до 21%;
- середня ймовірність класифікації моделі склала 83%, коливалась від 72% до 96%;
- кількість пройдених завдань в кожному сценарії в середньому склала 24.5 з 30 можливих, коливалась від 22 до 28;
- в середньому кожне тестування зайняло на 18.3% менше часу (через зменшену кількість необхідних завдань).

Рисунок Д.13 – Слайд презентації №13

Структурна схема розробленого програмного засобу

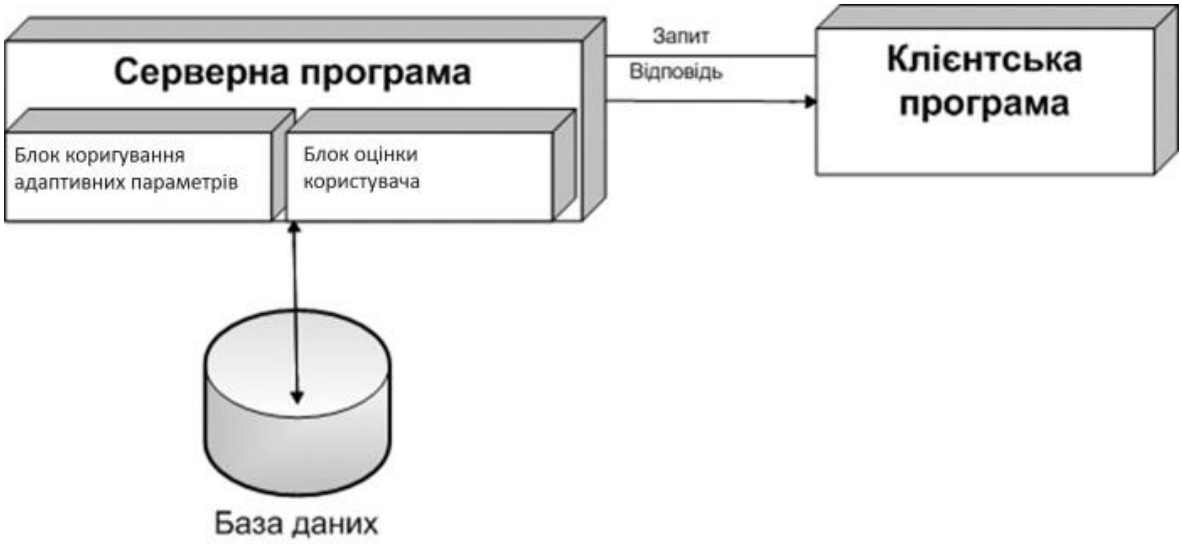


Рисунок Д.14 – Слайд презентації №14

Інтерфейс серверного додатку

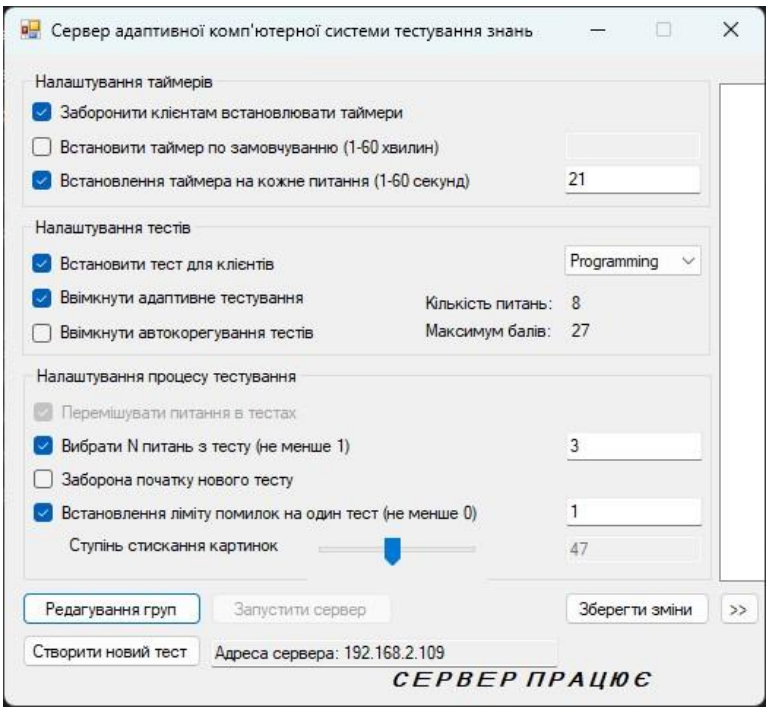


Рисунок Д.15 – Слайд презентації №15

Інтерфейс клієнтського додатку

Рисунок Д.16 – Слайд презентації №16

Наукова новизна та практичне значення

1. Розроблено метод **оцінки знань користувача** за допомогою моделі класифікаційного дерева, котра приймає рішення про **закінчення тестування** та визначає кінцевий результат тестування, що забезпечує зменшення часу процесу тестування та, відповідно, економію ресурсів для його забезпечення;
2. Подальшого розвитку отримав метод «адаптивності» комп'ютерного тестування знань за рахунок динамічного підбору коефіцієнтів зміни **складності завдань**, що дозволило більш точно формувати завдання за складністю, відповідно до рівня підготовки користувача, а також **калібрувати тестовізавдання**.

Рисунок Д.17 – Слайд презентації №17

Дякую за увагу!

Рисунок Д.18 – Слайд презентації №18