МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки та програмної інженерії
Кафедра інженерії програмного забезпечення

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
Катерина НЕСТЕРЕНКО
“____”_____2024 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСНИКА ОСВІТНЬОГО СТУПЕНЯ
“БАКАЛАВР”

Тема: “Застосунок розрахунку показника оцінки якості проекту на основі статичних даних”

Виконавець: Панасюк Олександр Сергійович

Керівник: к.т.н Клюєв Євгеній Іванович

Нормоконтролер: Варнавський В’ячеслав Володимирович

Київ 2024

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки та програмної інженерії
Кафедра інженерії програмного забезпечення
Освітній ступінь бакалавр
Спеціальність 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма «Програмне забезпечення систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Катерина НЕСТЕРЕНКО
"___" _____ 2024 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи студентки
Панасюка Олександра Сергійовича

1. Тема проекту: «Застосунок розрахунку показника оцінки якості проекту на основі статичних даних»
затверджена наказом ректора від 8.12.2024 р. № 2483/ст

2. Термін виконання проекту: з 3.01.2024 р. до 29.02.2024 р.

3. Вихідні дані до проекту: розробити програмний продукт який базуватиметься на статичних даних для розрахунку показника оцінки якості проекту

4. Зміст пояснювальної записки:
   1. Аналіз предметної області та огляд існуючих методологій та підходів до розрахунку показника якості проекту.
   2. Постановка завдання та мети дослідження. обґрунтування вибору розрахункового показника на основі статичних даних.
   3. Результати розробки застосунку.

5. Перелік обов'язкових слайдів презентації:
   1. Аналіз існуючих програм для оцінки якості проекту.
   2. Проблематика статичного оцінювання.
   3. Теорія нечітких множин.
   4. Формула Шортлліфа.
   5. Розробка власної методики оцінювання результативності проекту.
   6. Реалізація цієї методики в програму.

6. Календарний план-графік

№ пор	Завдання	Термін виконання	Відмітка про виконання
1.	Складання та затвердження графіку кваліфікаційної роботи Написання 1 розділу, представлення керівнику	19.01.24 – 23.01.24	
2.	Попередній друк 1 розділу та допоміжних сторінок (чорновик) - титульної, завдання, графіка, реферат, список скорочень, зміст, вступ, список джерел, 1-й нормо-контроль.	23.01.24 – 26.01.24	
3.	Написання 1 розділу, представлення керівнику	26.01.24 – 28.01.24	
4.	Написання 2 розділу, представлення керівнику	29.01.24 – 04.02.24	
5.	Написання 3 розділу, представлення керівнику	05.02.24 – 11.02.24	
6.	Загальне редагування та друк пояснювальної записки, графічного матеріалу	01.02.22 – 09.02.22	
7.	Проходження нормо-контролю, перевірка на антиплагіат, перепліт пояснювальної записки.	12.02.24 – 18.02.24	
8.	Розробка тексту доповіді. Оформлення графічного матеріалу для презентації	13.02.24 – 15.02.24	
9.	Отримання відгуку керівника, рецензії.	19.02.24 – 23.02.24	
10.	Підготовка матеріалів для передачі секретарю ДЕК (ПЗ, CD-R з електронними копіями ПЗ, презентації, відгук керівника, рецензія) в папці	26.02.24 – 28.02.24	

7. Дата видачі завдання 3.01.2022
Керівник: к.т.н. Євген КЛЮЄВ
Завдання прийняв до виконання: ПАНАСЮК ОЛЕКСАНДР
Дата

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Застосунок розрахунку показника оцінки якості проекту на основі статичних даних»: 52 с., 13 рис., 4 табл., 27 інформаційних джерел.

ОБРОБКА СТАТИЧНИХ ДАНИХ, ТЕОРІЯ НЕЧІТКИХ МНОЖИН, ФОРМУЛА ШОРТЛЛІФА.

Об’єкт розробки – застосунок, що автоматично обраховує оцінку якості проекту.

Мета роботи – підвищити ступінь автоматизації з оцінкою якості проекту на основі статичних даних.

ABSTRACT

Explanatory note to the qualification work "Application for calculating the project quality assessment indicator based on static data": 52 pages, 13 figures, 4 tables, 27 information sources.

PROCESSING OF STATIC DATA, THEORY OF FUZZY SETS, SHORTLIFFE'S FORMULA.

Property development – an application that automatically creates heat maps by analyzing the presented interface.

Purpose – increase the degree of automation in the work of interface designers.

ЗМІСТ

ПЕРЕЛІК ПРИЙНЯТИЙ СКОРОЧЕНЬ

ПЗ- програмне забезпечення

СУБД-система управління базами даних

GUI-Графічний інтерфейс користувача (*Graphical user interface*)

ВСТУП

Прогрес в сьогоднішньому світі не стоїть на місці, він вдосконалюється з кожною хвилиною, секундою, миттю. Щоб мати змогу не сповільняти, а навпаки- пришвидшувати прогрес створюються нові технології, удосконалюються, заміняються та виводяться з обрахунку старі.

В умовах активного технологічного розвитку та безперервного вдосконалення програмного забезпечення важливість якісних проектів має вирішальне значення. Високоякісне програмне забезпечення впливає на конкурентоспроможність компаній, ефективність роботи організацій та розвиток сучасного суспільства в цілому. Однак, існуючі методи оцінки якості проектів часто обмежуються динамічним аналізом без урахування важливих аспектів, пов'язаних зі статичними даними.

У зв'язку з цим існує потреба у подальшому розвитку та вдосконаленні методів оцінювання, зокрема у використанні статичних даних. До статичних даних відносяться характеристики програмного коду, архітектурні особливості проекту та інші постійні атрибути, які суттєво впливають на якість та супроводжуваність програмного продукту.

Ця дипломна робота присвячена розробці та застосуванню новаторського методу розрахунку показників оцінки якості проекту, базованого саме на аналізі статичних даних. Наш підхід передбачає глибокий аналіз якісних та кількісних аспектів програмного коду, структури проекту та інших фіксованих параметрів, з метою створення засобу об'єктивної та комплексної оцінки якості проектів ще на етапі їх проектування та створення.

Очікується, що результати цього дослідження матимуть значний вплив на розробку програмного забезпечення та управління проектами і сприятимуть підвищенню стандартів якості, операційної ефективності та стабільності в галузі інформаційних технологій та розробки програмного забезпечення.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОГЛЯД ІСНУЮЧИХ МЕТОДОЛОГІЙ ТА ПІДХОДІВ ДО РОЗРАХУНКУ ПОКАЗНИКА ЯКОСТІ ПРОЕКТУ

Аналіз попередніх досліджень та методів оцінки якості проектів визначає важливість та актуальність цієї теми в контексті сучасної розробки програмного забезпечення. Детальний огляд літератури та наукових джерел дозволяє виявити існуючі підходи та заповнити прогалини новими методами оцінки якості проектів на основі статичних даних:

1.1. Статичний аналіз коду.

1.2. Аналіз архітектурних рішень.

1.3. Використання спеціалізованих інструментів.

1.4. Існуючі методології.

1.5. Статичні методи оцінювання.

1.6. Проблематика статичного оцінювання.

На основі цих методик я буду проводити аналіз досліджень які вже вийшли в маси та широко використовуються для ‘відсіювання’ або ж просування проектів на ранніх етапах.

1.1. Метод статичного аналізу(*Static Analysis*)

Статичний аналіз коду (також відомий як аналіз вихідного коду) зазвичай виконується як частина верифікації коду (також відома як тестування білого ящика) і здійснюється на етапі реалізації життєвого циклу розробки безпеки (*SDL*). Статичний аналіз коду зазвичай означає запуск інструменту статичного аналізу коду. Інструменти статичного аналізу коду намагаються виявити потенційні вразливості в "статичному" (невиконуваному) вихідному коді за допомогою таких методів, як аналіз дефектів і аналіз потоків даних.

В ідеалі, такі інструменти повинні виявляти недоліки безпеки автоматично і з високим ступенем впевненості в тому, що знайдене дійсно є недоліком. Однак для багатьох типів вразливостей додатків це виходить за рамки сучасних технологій. Тому замість того, щоб бути просто інструментом, який автоматично виявляє вразливості, такі інструменти діють як допоміжний інструмент, який допомагає аналітикам зосередитися на важливих для безпеки частинах коду, щоб вони могли виявляти вразливості більш ефективно.

Деякі інструменти почали переходити до інтегрованих середовищ розробки (*IDE*). Для тих типів проблем, які можна виявити на етапі розробки програмного забезпечення, ефективніше використовувати такі інструменти на цьому етапі життєвого циклу розробки. Такий негайний зворотній зв'язок набагато корисніший, ніж пошук вразливостей на більш пізніх етапах циклу розробки.

Британський оборонний стандарт 00-55 вимагає використання статичного аналізу коду для всього "програмного забезпечення, пов'язаного з безпекою оборонного обладнання"(рис.1.1.1).

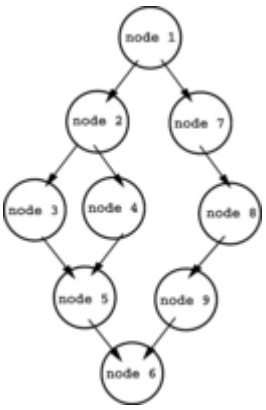


рис.1.1.1

Більш детально вивчивши метод статичного аналізу можна зробити висновок, що він виступає важливим етапом у розробці програмного забезпечення, що полягає в досліджені оцінки програмного коду без його фактичного виконання.

Розглянувши основні аспекти статичного аналізу коду можна виділити основні з них, такі як:

- Метрика якості коду.
- Аналіз синтаксису.
- Виявлення помилок та аномалій.
- Пошук кодових ‘запахів’.

Метрика якості коду.

Метрики якості коду використовуються для об'єктивного оцінювання різних аспектів програмного коду з метою покращення читабельності, ефективності, ремонтопридатності та надійності. Важливо пам'ятати, що жодна метрика не є панацеєю і повинна використовуватися для конкретного проекту і контексту. Ось деякі з ключових метрик якості коду:

Кількість рядків коду(*Lines Of Code- LOC*).

Метрика Lines of Code (*LOC*) - одна з основних метрик якості коду, яка визначає загальну кількість рядків коду в проекті. Ця метрика дуже проста, але надає корисну інформацію про розмір, обсяг і складність кодової бази. Важливо

зазначити, що кількість рядків коду сама по собі не є точним показником якості або ефективності коду.

Ключові аспекти метрики кількості рядків коду:

Розмір проекту: кількість рядків коду є мірою обсягу роботи, необхідної для проекту. Загалом, більша кількість рядків коду вказує на те, що додаток є більш складним і великим.

Складність кодової бази: загальна кількість рядків коду є показником складності системи. Однак для більш повного уявлення про складність програми важливо оцінювати цей показник у поєднанні з іншими показниками, такими як циклічна складність.

Простота підтримки та обслуговування: великі обсяги коду складніше модифікувати та розвивати. Зменшення кількості рядків коду шляхом ефективної організації та видалення непотрібного коду полегшує підтримку проектів.

Історія змін та відстеження:кількість рядків коду є індикатором для відстеження змін у розвитку проекту. Збільшення кількості рядків вказує на вдосконалення та розширення, тоді як зменшення кількості рядків означає оптимізацію та рефакторинг.

Оцінка продуктивності розробника: рядки коду також можна використовувати для вимірювання продуктивності розробника. Однак, слід бути обережним у цій оцінці, оскільки більша кількість рядків коду не обов'язково означає кращу продуктивність.

Використовуючи метрику рядків коду, важливо враховувати контекст і специфіку проекту. Оцінка якості та продуктивності коду повинна також включати аналіз інших метрик та загальної архітектури проекту.

Цикломатична складність(*Cyclomatic Complexity*)

Циклічна складність - це показник якості коду, який визначає кількість незалежних шляхів, що можуть бути виконані в програмі. Ця метрика була вперше запропонована Томасом Маккейбом у 1976 році і є одним з основних показників, що використовуються для визначення складності коду.

Основні аспекти цикломатичної складності:

- Контрольні точки: цикломатична складність визначається кількістю контрольних точок у програмі. Контрольні точки - це місця в коді, де реалізується управління потоком, наприклад, умовні рішення, цикли та виклики функцій.
- Потенційні шляхи: кожна контрольна точка визначає потенційний шлях, який може виконати програма. Циклічна складність відображає кількість таких потенційних шляхів.
- Формула МакКейба: циклічну складність можна обчислити за формулою МакКейба (рис1.1.2):

$$M = E - N + 2P \quad (1.1.2)$$

Де (E) - кількість ребер (зв'язків), (N) - кількість вузлів (контрольних точок) і (P) - кількість з'єднаних компонентів (з'єднаних областей) у графі потоку управління.

- Інтерпретація значень: значення цикломатичної складності часто використовують для оцінки потенційної складності тестування та супроводу коду. Висока цикломатична складність вказує на те, що код складний для розуміння та тестування.
- Покриття тестами: це пов'язано з тим, що всі незалежні шляхи в коді повинні бути покриті, щоб гарантувати всебічне тестування.
- Рефакторинг: рефакторинг коду може зменшити циклічну складність.

Цикломатична складність може допомогти розробникам приймати рішення щодо тестування та організації коду, але важливо оцінювати її в контексті інших метрик та характеристик проекту.

Індекс читабельності коду(*Code Readability Index*)

Індекс читабельності коду - це показник для визначення рівня читабельності та зрозумілості програмного коду. Для вимірювання читабельності коду розроблені різні інструменти та методи, і індекс читабельності є одним з них.

Основні аспекти індексу читабельності коду:

- Довжина рядків коду: одним з аспектів визначення читабельності є довжина рядків коду. Загалом, коротші рядки коду легше зрозуміти з першого погляду і тому вони вважаються більш читабельними.
- Термінологічна вкладеність і складність: читабельність також залежить від кількості вкладеності та складності конструкцій, таких як умовні оператори та цикли. Чим менше вкладеності та складності, тим легше зрозуміти код.
- Кількість параметрів і змінних: кількість параметрів і змінних впливає на читабельність коду. Занадто велика кількість параметрів і змінних ускладнює розуміння коду.
- Коментарі та документація: це особливо актуально, якщо іншим розробникам потрібно зрозуміти логіку.
- Використовуйте імена, які легко зрозуміти: назви змінних, функцій і класів повинні бути описовими і зрозумілими. Вони мають важливий вплив на враження від коду.
- Усунення зайвих деталей: непотрібні деталі та складні структури, що не несуть значущої інформації, роблять код важким для читання та розуміння.

Показники читабельності коду можна визначити за допомогою різних формул і методів, наприклад, індексу читабельності Флеша-Кінкейда. Використання цього індексу полегшує розробникам вибір найкращих практик при написанні коду, роблячи код легшим для розуміння та підтримки.

Кількість дефектів на одиницю коду(*Defects per KLOC*)

"*Bugs per KLOC*" - це показник якості коду, який показує кількість виявлених помилок або дефектів на 1000 рядків коду. Цей показник використовується для оцінки якості програмного продукту та ефективності процесу розробки.

Ключові аспекти "дефектів на одиницю коду:

- Визначення дефектів:

Дефекти або помилки включають невідповідність вимогам, некоректну роботу програми або інші проблеми, які можуть виникнути під час експлуатації програмного продукту.

- Кількість дефектів:
  - Цей показник враховує загальну кількість дефектів, знайдених протягом певного періоду або фази розробки.
  - Нормалізація за розміром коду (*KLOC*): для об'єктивного порівняння кількість дефектів на одиницю коду нормалізується за розміром коду, тобто за кількістю тисяч рядків коду (*KLOC*). Це гарантує, що розмір проекту враховується при оцінці якості проекту.
  - Оцінка якості коду: цей показник надає інформацію про якість коду. Чим менше помилок на одиницю коду, тим більша ймовірність того, що додаток працює правильно і містить менше помилок.
  - Відстеження змін: слід відстежувати зміни кількості помилок на одиницю коду з плином часу. Збільшення цього показника може свідчити про проблеми в процесі розробки або погіршення якості продукту.
  - План тестування: значення цієї метрики дозволяє команді розробників планувати тести, збільшуючи або зменшуючи їх інтенсивність залежно від того, наскільки важливо досягти високого рівня якості коду.

Ця метрика допомагає команді розробників визначити ефективність процесу і забезпечити високий рівень якості коду.

Ступінь зчеплення та Когезія(*Coupling and Cohesion*)

Ступінь зчеплення та когезія - це дві ключові концепції в архітектурі програмного забезпечення, які визначають взаємозв'язок та внутрішню організацію компонентів системи.

Ступінь зчеплення (*Coupling*):

Ступінь зчеплення визначає ступінь взаємозалежності між різними компонентами або модулями системи. Це означає, наскільки один модуль залежить від інших. Чим нижчий рівень зчеплення, тим менше взаємозалежності, і тим ефективніше можна модифікувати окремі компоненти.

Типи ступенів зчеплення:

- Слабке (*Low*): компоненти мають мінімальні взаємозалежності.
- Середнє (*Moderate*): є деякі взаємозалежності, але вони керуються чітким інтерфейсом.
- Сильне (*High*): велика взаємозалежність, зміни в одному компоненті можуть впливати на інші.

Зниження ступеня зчеплення може полегшити розвиток, тестування та утримання системи, оскільки зміни в одному компоненті не впливатимуть безпосередньо на інші.

Когезія (*Cohesion*):

Когезія визначає, наскільки сильно пов'язані функції або обов'язки одного компонента. Це вказує на те, наскільки логічно та однорідно організовані функції в межах компонента. Чим вищий рівень когезії, тим більш зв'язані функції в компоненті.

Типи когезії:

- Функційна (*Functional*): компонент виконує одну конкретну функцію або обов'язок.
- Логічна (*Logical*): компонент має схожі функції, які логічно групуються.
- Часова (*Temporal*): компоненти використовуються в один і той же час.
- Просторова (*Spatial*): компоненти розташовані в одній зоні пам'яті або просторово близькі.

Висока когезія означає, що функції в компоненті пов'язані і спільно виконують якусь конкретну задачу.

Правильне співвідношення ступеня зчеплення та когезії є ключем до ефективної архітектури програмного забезпечення. Велика когезія та низький рівень зчеплення допомагають покращити розширюваність, тестованість та обслуговуваність системи.

Метрика забезпечення безпеки.

Метрика забезпечення безпеки - це система числових показників, яка використовується для вимірювання та оцінки ефективності заходів безпеки в

інформаційних системах. Ці метрики дозволяють організаціям та командам забезпечення безпеки отримувати об'єктивну інформацію про рівень захищеності, ідентифікувати слабкі місця та вдосконалювати стратегії безпеки.

Основні аспекти метрик забезпечення безпеки:

- Кількісні метрики:
  - Кількість виявлених вразливостей: визначає кількість потенційних слабкі місць або дірок у захисті системи.
  - Кількість інцидентів безпеки: ілюструє обсяг подій, пов'язаних із порушенням безпеки, які сталися в системі.
- Якісні метрики:
  - Час виявлення та відновлення (*Time to Detect, Time to Recover*): визначає, як швидко система виявляє порушення безпеки та відновлюється після інциденту.
  - Рівень ефективності заходів безпеки: оцінює, наскільки ефективні заходи безпеки у запобіганні атакам та мінімізації можливих шкідливих наслідків.
- Метрики вартості:
  - Вартість впровадження та підтримки заходів безпеки: визначає фінансові витрати на заходи безпеки та їхнє утримання.
  - Вартість втрат внаслідок інцидентів безпеки: оцінює фінансові збитки, пов'язані з інцидентами безпеки, включаючи втрату даних, зниження продуктивності та репутаційні втрати.

Ці метрики є інструментами для визначення, контролю та вдосконалення безпеки інформаційної системи та допомагають забезпечувати відповідність стандартам безпеки та найкращим практикам.

Аналіз синтаксису.

Аналіз синтаксису - це процес вивчення та розуміння структури речень чи коду в мові програмування. У залежності від контексту, аналіз синтаксису може відноситися як до області мовознавства, де досліджується граматика

різних мов, так і до програмування, де визначається правильна структура вихідного коду.

- Мовознавство: у лінгвістиці, аналіз синтаксису займається граматичною структурою речень та фраз в мові. Вивчення синтаксису допомагає розуміти, як слова та фрази комбінуються для створення змістовного мовлення. В аналізі синтаксису важливо визначити правила, які визначають коректну граматичну структуру речень.
- Програмування: у програмуванні аналіз синтаксису стосується визначення правил, які визначають правильну структуру програмного коду. Це включає в себе визначення правильної послідовності ключових слів, операторів та інших елементів мови програмування. Аналіз синтаксису допомагає компіляторам або інтерпретаторам перетворювати вихідний код на рівень, який може бути виконаний комп'ютером.
- Лексичний та синтаксичний аналізатори: для виконання аналізу синтаксису в програмуванні використовуються лексичні та синтаксичні аналізатори. Лексичний аналізатор розбиває вихідний код на токени (лексеми), тоді як синтаксичний аналізатор визначає, як ці токени комбінуються в речення чи програмну конструкцію згідно з граматикою мови.
- Помилки синтаксису: помилки синтаксису в програмному коді вказують на те, що код не відповідає правилам мови програмування. Інструменти редагування коду та компілятори зазвичай виводять повідомлення про помилку синтаксису, щоб розробники могли виправити невірно введений або некоректний код.

Аналіз синтаксису важливий в обох контекстах, оскільки він допомагає забезпечити правильність мовлення в мовознавстві та визначає правильність конструкцій у програмуванні, забезпечуючи тим самим зрозумілість та коректність обробки інформації.

Виявлення помилок та аномалій.

Виявлення помилок та аномалій є важливим етапом в розробці програмного забезпечення. Цей процес спрямований на ідентифікацію та усунення неправильностей, що можуть виникнути під час розробки або експлуатації системи. Розглянемо цей процес більш детально:

- Визначення помилок та аномалій:
- Помилки (*Errors*): це неправильності у вихідному коді або дизайні програми, які призводять до некоректної роботи системи. Помилки можуть виникнути через невірне введення, неправильні алгоритми чи синтаксичні невідповідності.
- Аномалії (*Anomalies*): це відхилення від очікуваного або нормального стану програми. Аномалії можуть включати неправильності в даних, виявлені під час виконання програми.
- Локалізація помилок:
- Важливо визначити точне місце в коді або дизайні, де виникає помилка. Це може вимагати використання інструментів для налагодження (*debugging tools*), логування (*logging*), а також аналізу стеку викликів (*stack trace*) для встановлення точного місця виникнення помилки.
- Відділення помилок та відлагодження:
- Виявлені помилки можуть бути різного характеру, від синтаксичних помилок до логічних. Для виправлення помилок можуть використовуватися інструменти відлагодження, які дозволяють розробникам крок за кроком виконувати код та спостерігати за його станом.

Виявлення помилок та аномалій є невід'ємною частиною циклу розробки програмного забезпечення. Ефективне управління цим процесом дозволяє створювати високоякісне програмне забезпечення, яке відповідає вимогам та очікуванням користувачів.

Пошук кодових ‘запахів’.

Виявлення кодового диму - це процес виявлення потенційних проблем у програмному коді, які можуть призвести до збільшення складності, зниження

зрозумілості та погіршення якості і продуктивності коду. Кодовий дим може вказувати на те, що код можна покращити або оптимізувати. Основна ідея полягає у виявленні "запахів", які можуть вказувати на проблеми в архітектурі, дизайні або використанні мови програмування.

Розгляну деякі поширені "запахи" коду та способи їх виявлення:

- Повторення коду:

- Запах: повторення схожого коду в різних частинах програми.
- Пошук: використовуйте такі інструменти, як інструменти виявлення клонів, щоб проаналізувати повторення коду.

- Довгий метод:

- Визначити: метод із надто великою кількістю рядків коду.
- Пошук: аналіз розміру методу під час перегляду коду або використання таких критеріїв, як кількість рядків коду в методі.

- Довгі сигнатури методів (довгі списки параметрів):

- Визначення: метод, який отримує велику кількість параметрів.
- Як шукати: перегляд сигнатур методів для виявлення надмірної кількості параметрів.

Рішення: використовуйте метрики для визначення кількості класів та їхньої структури.

Погані коментарі:

Визначення: коментарі, які є неточними або містять застарілу інформацію.

Рішення: переглядайте коментарі під час перегляду коду та аналізуйте їх релевантність.

Забагато локальних змінних:

- Визначення: метод, який використовує велику кількість локальних змінних: метод, який використовує велику кількість локальних змінних.
- Пошук: використання метрик для визначення кількості та обсягу локальних змінних у методі.

- Використання інструментів статичного аналізу коду та проведення регулярних оглядів і аудитів коду під час розробки може допомогти знайти "запахи" в коді. Виявлення та виправлення "запахів" коду може допомогти підвищити якість і полегшити підтримку програмного забезпечення.

1.1. АНАЛІЗ АРХІТЕКТУРНИХ РІШЕНЬ

Аналіз архітектурних рішень - це складний процес, який включає детальне дослідження та оцінку можливих структур і концепцій побудови програмного забезпечення. Аналітики обговорюють і визначають потенційні архітектурні сценарії, починаючи з розуміння системних вимог, включаючи функціональні та нефункціональні аспекти.

У цьому процесі важливо оцінити ступінь інноваційності обраної технології та її вплив на продуктивність і масштабованість системи. Також аналізуються питання безпеки та конфіденційності, визначаються потенційні ризики та розробляються стратегії захисту.

Гнучкість і масштабованість архітектури також будуть розглянуті, оскільки важливо, щоб система могла легко адаптуватися до змін і розширень. Взаємодія та інтеграція з існуючими системами та послугами також будуть ретельно розглянуті.

Ще одним аспектом є вартість розробки та впровадження системи. Аналіз витрат дозволяє визначити фінансові аспекти обраної архітектури та, за необхідності, розробити план переходу.

У процесі моделювання та проектування архітектор представляє взаємодію компонентів і структуру системи у вигляді архітектурних моделей і діаграм.

Заключним етапом є оцінка ризиків, пов'язаних з обраною архітектурою, та розробка стратегій управління цими ризиками. Розробка плану переходу є ключовим елементом для безперешкодного впровадження обраної архітектури.

1.2. ВИКОРИСТАННЯ СПЕЦІАЛІЗОВАНИХ ІНСТРУМЕНТІВ

Використання спеціальних інструментів в процесі аналізу архітектурних рішень є важливим елементом для ефективного і точного визначення найбільш підходящого підходу до створення програмного забезпечення.

Ці інструменти дозволяють аналітикам і архітекторам детально вивчити параметри, що впливають на обране будівельне рішення. Використання цих інструментів спрощує збір та обробку великих обсягів даних і допомагає приймати обґрунтовані рішення.

Сюди відносяться інструменти для аналізу коду з точки зору ефективності, безпеки та виявлення можливих дефектів. Інструменти статичного аналізу також допомагають оцінити якість коду і виявити "запахи" коду, які можуть свідчити про потенційні проблеми.

Існують також інструменти для аналізу вимог, які можуть враховувати різні аспекти функціональних і нефункціональних потреб. Це полегшує вибір архітектурних рішень, які найкраще відповідають конкретним вимогам.

Інструменти моделювання та візуалізації дозволяють створювати архітектурні моделі та діаграми для кращого розуміння взаємодії компонентів системи та її структури.

Оцінка вартості та продуктивності здійснюється за допомогою інструментів аналізу вартості та продуктивності, які допомагають визначити ефективність та вартість розробки та експлуатації.

Таким чином, використання спеціалізованих інструментів є необхідною складовою успішного аналізу архітектурних рішень, забезпечуючи точність, об'єктивність і швидкість в процесі прийняття важливих стратегічних рішень.

1.3. **ІСНУЮЧІ МЕТОДОЛОГІЇ**

Існують різні методології та підходи до оцінки якості проектів, кожен з яких має свої особливості.

Однією з них є *PMBOK*, яка визначає стандарти і процеси для ефективного управління проектами. Він розглядає планування, контроль і забезпечення якості як основні елементи; *PMBOK* використовує ключові показники ефективності (*KPI*) для вимірювання різних аспектів якості.

Six Sigma - це методологія, спрямована на підвищення якості та ефективності процесів. Вона використовує цикл *DMAIC* для постійного вдосконалення, включаючи вимірювання рівня помилок та ефективності процесу.

Методології *Agile* та *Scrum* наголошують на ітеративному підході до розробки та управління проектами і використовують такі показники, як регулярне тестування та швидкість розробки.

ISO 9000 - це набір стандартів, які визначають принципи систем управління якістю, включаючи аудит та відповідність.

Kanban фокусується на візуалізації та оптимізації робочих процесів за допомогою таких показників, як тривалість циклу виконання завдань.

Інтеграція моделі зрілості спроможностей (*Capability Maturity Model Integration, CMMI)* забезпечує основу для оцінки та вдосконалення процесів за допомогою набору ключових показників.

Кожен з цих підходів має власні методи та інструменти для розрахунку якості проекту, націленість на ефективність та високу задоволеність результатами.

1.4. СТАТИЧНІ МЕТОДИ ОЦІНЮВАННЯ ПРОЕКТУ

В сфері оцінювання якості проекту вже існує маса програмних засобів для його оцінки та побудови графіків. Кожен з них базується на своїй мові програмування, вони включають в себе різні функції, методи, теорії для визначення певних вимог, для вирішення та відображення потрібних задач.

Одним з таких являється інструмент *Maple*. *Maple* – це комерційна система комп'ютерної алгебри. Вона включає більшість розділів сучасної математики, понад 5000 функцій для моделювання та інтерактивної візуалізації, підтримує мову програмування *Maple* і дозволяє створювати електронні документи, що містять алгоритми, результати обчислень, математичні вирази, текст, графіки, діаграми та звукову анімацію. Можливості пакету - символьні обчислення та чисельні методи; математичні функції та методи; розв'язування рівнянь; диференціальні рівняння; лінійна алгебра; оптимізація; програмування; маніпулювання розмірностями та одиницями вимірювання; редактор формул; візуалізація, графіка, інтерактивні меню та помічники; зразки шаблонів для стандартних задач. Елементи для розробки графічних інтерфейсів; доступ до сховища *MapleCloud* для обміну документами між користувачами та колегами; понад 30 патернів для створення та редагування формул; розпізнавання рукописних формул, фінансове моделювання, статистичне моделювання, фізичне моделювання, високопродуктивні обчислення, автоматичне розпаралелювання, багатопотокове програмування, обчислення в грід-мережах, підтримка CUDA, інтерфейс до Matlab, експорт на інші мови програмування,

система доступу до баз даних, інтерфейс до математичної бібліотеки NAG(рис. 1.5.1).

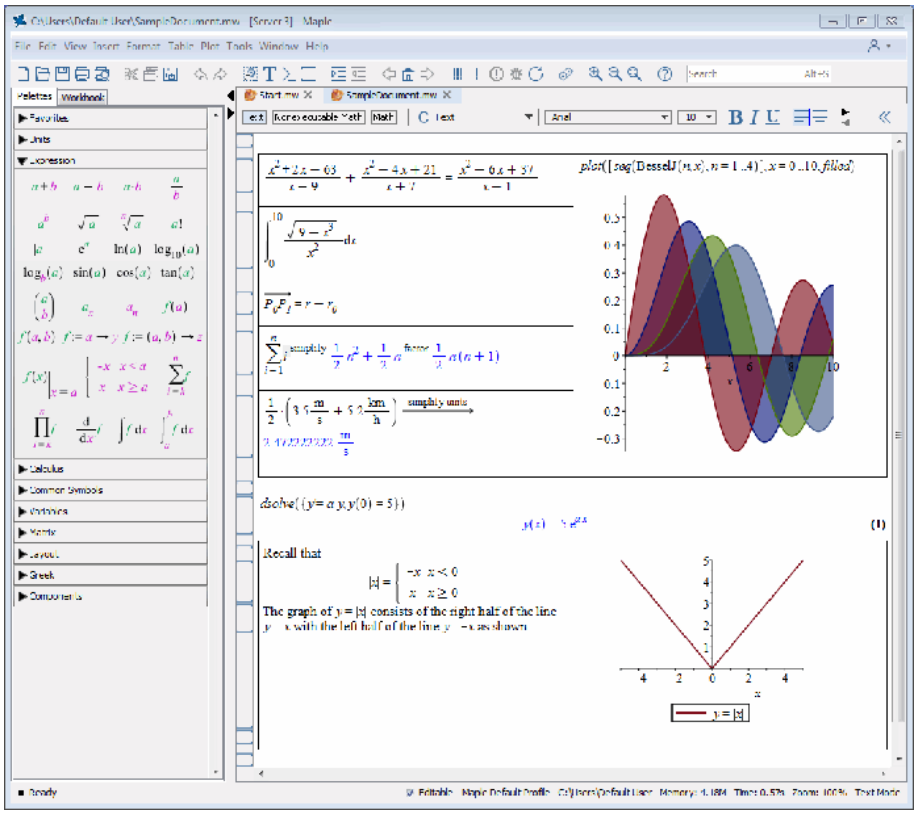


рис.1.5.1

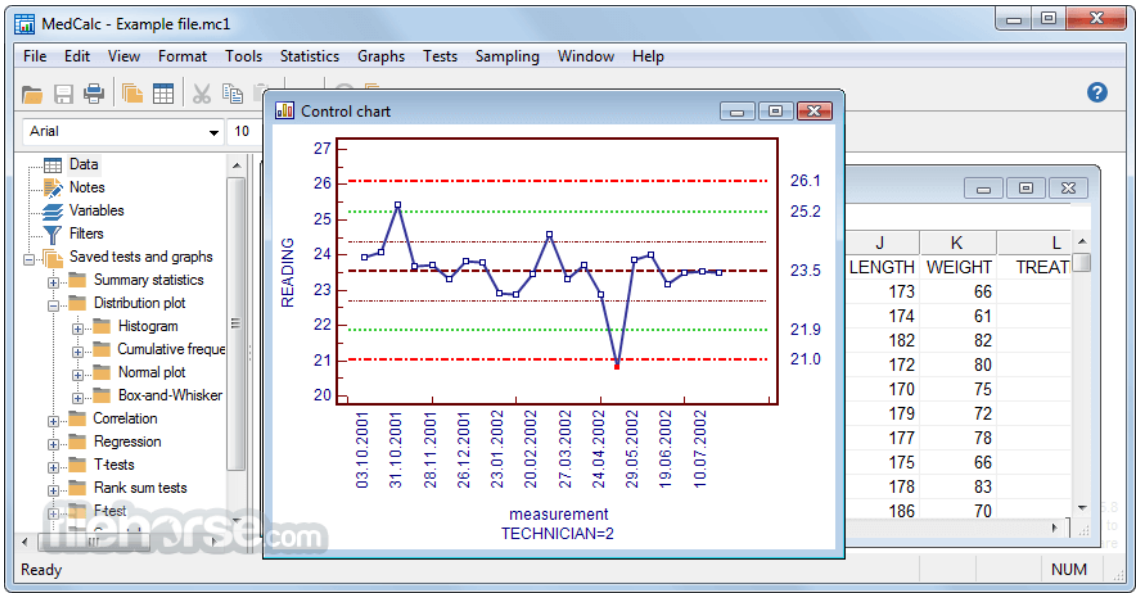


рис.1.5.2

MedCalc – невелика програма для аналізу статистичних даних. Вона дуже маленька, містить лише найнеобхідніші функції при розрахунках біологічних та медичних експериментів, реалізує функції аналізу *ROC*-кривих(рис.1.5.2).

GraphPad Prism - програма, що спеціалізується на статистичному аналізі біологічних даних (біостатистика, криві, графіки). *GraphPad Prism* також можна використовувати для конвертації файлів між різними форматами. *GraphPad Prism* поєднує в собі можливість створювати наукові графіки, генерувати криві для нелінійної регресії, отримувати чіткі статистичні результати та організовувати дані. Розроблена для біологів, соціологів та фізиків, ця програма також широко використовується студентами та аспірантами. Нелінійна регресія є важливим інструментом в аналізі даних, але її завдання часто буває складнішим, ніж повинно бути; в *GraphPad Prism* побудова кривих спрощена до межі. Вам потрібно лише вибрати з великого списку часто використовуваних рівнянь, і програма зробить все інше автоматично - генерацію кривих, відображення результатів у таблицях, нанесення кривих на графіки, інтерполяцію невідомих значень і т.д. Система дозволяє легко виконувати основні статистичні тести, які зазвичай використовуються в лабораторних і клінічних дослідженнях: t-тести, непараметричні порівняння, однофакторний дисперсійний аналіз, двофакторний дисперсійний аналіз, аналіз таблиць умовних значень і аналіз виживаності. Всі частини проекту, що виконуються програмою, взаємопов'язані між собою, тому помилки в даних можна виправити один раз і отримати автоматично оновлені результати(рис.1.5.3.).

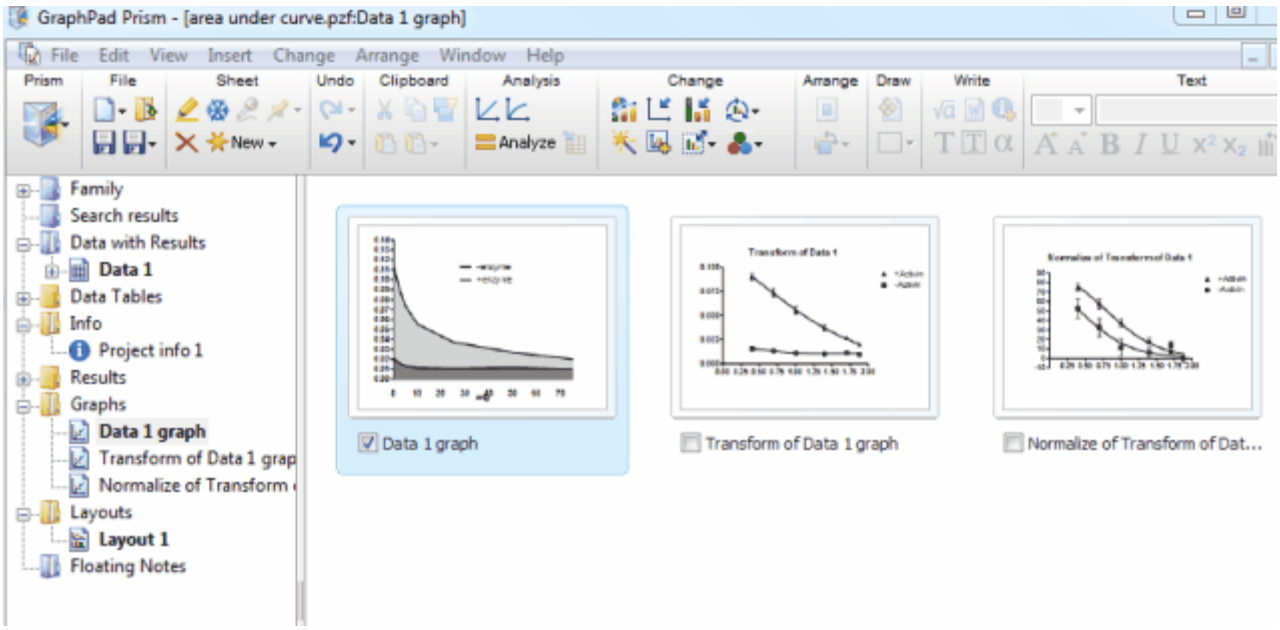


Рис.1.5.3.

1.5. ПРОБЛЕМАТИКА СТАТИНОГО ОЦІНЮВАННЯ ПРОЕКТУ

Оцінка якості проекту на основі статичних даних має обмеження та недоліки. По-перше, цей підхід повинен враховувати, що код аналізується статично в певний момент часу, без урахування динамічних змін під час виконання. Це обмеження ускладнює оцінку деяких аспектів якості.

Однією з головних проблем є те, що воно обмежує вимірювання функціональності. Статичний аналіз часто обмежується визначенням конкретних аспектів, таких як структура коду, метрики якості коду та статичні аналізатори. Це може не враховувати адекватно різні аспекти функціональності, ефективності та надійності програмного забезпечення.

Недоліком також є неадекватна інтерпретація контексту використання. Статичний аналіз може не враховувати конкретні умови використання програмного продукту або його взаємодію з іншими компонентами і системами, що обмежує можливість точного визначення важливих аспектів якості.

Статичному аналізу важко враховувати відгуки користувачів та зміни у вимогах. Статичний аналіз не надає механізмів для опису фактичного користувацького досвіду або врахування змін у вимогах, які можуть виникнути під час експлуатації продукту.

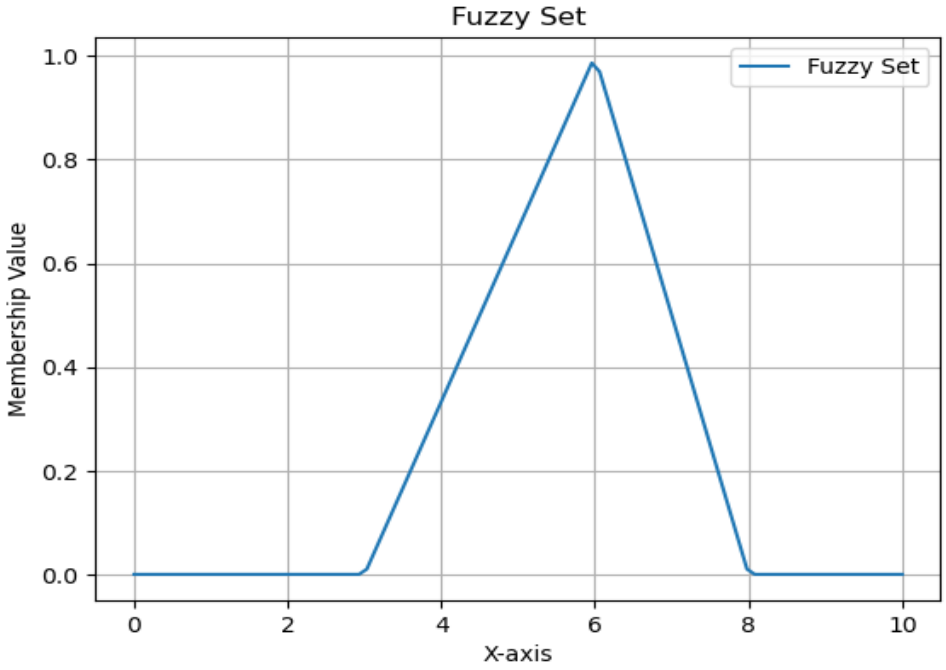
Слід також зазначити, що статичний аналіз не завжди враховує бізнес-контекст і стратегічні цілі компанії, які є важливими для оцінки успішності проекту. Наприклад, він може неадекватно враховувати фактори, що впливають на ринкову конкурентоспроможність або виробничі витрати.

Для великих, складних систем статичний аналіз може бути надто обтяжливим, а великий обсяг коду і кількість взаємодіючих компонентів можуть знизити точність результатів аналізу. Однак, незважаючи на ці недоліки, статичний аналіз залишається важливим компонентом оцінки якості проекту, особливо на ранніх стадіях розробки. Його найкраще використовувати в поєднанні з іншими методами та підходами для отримання повної картини якості програмного забезпечення.

В основу мого дипломного проекту буде покладено суміш динамічного с статичним методом оцінювання, оскільки статичний метод потребує обширної бази даних, з яких можна брати інформацію і це слугуватиме більш якісною оцінкою, в моєму проекту, можна буде використовувати невеликий об’єм даних і на експертних висловлюванях та апарату нечітких множин можна буде більш ретельно оцінити проект і при невеликому за об’ємом масивом даних цей підрахунок можна буде вважати більш точним та прийнятним.

Для прикладу, якщо ми візьмемо проект стаціонарних кавових машин самообслуговування які будуть розташовані на вхідній групі в будівлю. Для оцінки в нас буде 3 експерта, яким буде запропоновано оцінити від 1 до 10, якість приготування кави, серед інших аналогічних автоматів, прогнозована оцінка замовника буде 6, оскільки сам замовник є великим гурманом кави, та перекуштував каву в багатьох закладах, і його оцінювання буде базуватися на

цьому. Отже припустимо, що 3 експерта дали оцінку якості кави таку: 1 експерт оцінив на 3 бали, 2-й дав оцінку в 6,а 3-тьому вона дуже сильно сподобалася і він оцінив її в 8 балів.



(рис.1.6)

Середнє арифметичне даних оцінок буде наближена до 5.73. Якщо побудувати графік для заданого завдання за допомогою теорії нечітких множин вигляд дана функція буде мати(рис.1.6).

Висновки

Аналіз попередніх досліджень та методів оцінки якості проектів відображає важливість та актуальність цієї теми в сучасній розробці програмного забезпечення. Детальний огляд літератури та наукових джерел дозволяє виявити існуючі підходи та заповнити прогалини новими методами оцінки якості проектів на основі статичних даних.

Статичний аналіз коду, аналіз архітектурних рішень та використання спеціалізованих інструментів є основними методами, що використовуються для оцінки якості програмного забезпечення. Крім того, існують різні методології,

спрямовані на покращення процесу розробки та оцінки якості програмних продуктів.

Особлива увага приділяється статичним методам оцінювання, які базуються на аналізі програмних текстів та структур. Проте існують певні проблеми з цим підходом, такі як складність аналізу великих обсягів коду та нечіткість результатів.

На основі цих методик проводиться аналіз досліджень, що вже вийшли в маси та широко використовуються для відсіювання або просування проектів на ранніх етапах. Оцінка цих досліджень дозволить визначити найбільш ефективні та перспективні підходи до оцінки якості проектів програмного забезпечення.

РОЗДІЛ 2

ПОСТАНОВКА ЗАВДАННЯ ТА МЕТИ ДОСЛІДЖЕННЯ. ОБГРУНТУВАННЯ ВИБОРУ РОЗРАХУНКОВОГО ПОКАЗНИКА НА ОСНОВІ СТАТИЧНИХ ДАНИХ

Мета роботи: розробити програмний застосунок, який використовує теорію нечітких множин для розрахунку оцінки якості проекту на основі статичних даних, представлених експертами у графічній формі.

Завдання роботи:

2.1. Розробка інтерфейсу для представлення нечітких множин:
- Розробити зручний інтерфейс, де експерти можуть графічно виразити свої висловлювання в межах від 0 до 1, використовуючи теорію нечітких множин.

2.2. Розробка алгоритму обробки графічних даних:
- Розробити алгоритм, який перетворює графічно представлені нечіткі множини в числові дані в межах від 0 до 1.

2.3. Впровадження формули Шортліфа для розрахунку оцінки якості проекту:
- Імплементувати формулу Шортліфа для обчислення оцінки якості проекту на основі отриманих числових даних.

Очікувані результати: очікується, що розроблений застосунок буде здатний ефективно обробляти графічні висловлювання експертів, використовуючи теорію нечітких множин, і забезпечить об'єктивну оцінку якості проектів на основі статичних даних.

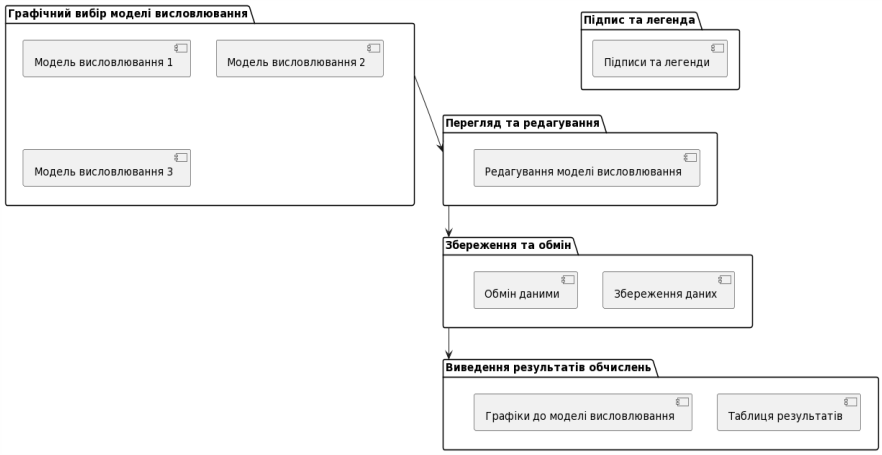
Методологія: у роботі буде використано комбінацію методів програмування та математичного моделювання для досягнення поставлених завдань.

2.1. Розробка інтерфейсу для представлення нечітких множин.

Обраний мною метод може працювати з малим об’ємом інформації. Експерти які будуть обиратися під оцінку, мають бути ознайомлені з ДСТУ оцінювання якості проекту, мають мати представлення про нечіткі множини і виявлення думки графічними методами.Інтерфейс програми буде містити:

- Графічний вибір моделі висловлювання. Експерт зможе за допомогою вибору малюнка і підставлення необхідних даних показати свою модель висловлювання.
- Підпис та легенда. Буде підбиратися певна група експертів, під свою задачу і буде малюватися відповідний графік з підписами та легендами.
- Перегляд та редагування. Користувач зможе додавати або ж видозмінювати свою модель висловлювання відповідно до своєї манери бачення.
- Збереження та обмін. Оцінка якості проекту і всі дані які були введені, будуть зберігатися і за рахунок цього, можна буде обмінюватися інформацією між експертами та замовником
- Виведення результатів обчислень. Всі дані які будуть обраховуватися будуть виводитися в вигляді таблиці в результуючому рядку і також будуть виводитися графіки до моделі висловлювання.

Я за допомогою діаграми компонентів графічно зообразив компоненти і зв’язки між ними(рис2.1.1).



(рис2.1.1)

Графічний вибір моделі висловлювання може бути важливим інструментом для експертів, що працюють з нечіткими множинами. Цей інтерфейс може включати в себе графічні символи або малюнки, які дозволяють експертам візуально представляти свої моделі висловлювань. За допомогою цього інтерфейсу експерти можуть вибирати різні графічні символи або малюнки, що краще відповідають їхнім моделям висловлювань, та додавати необхідні дані для цих моделей. Наприклад, експерт може вибрати обмеження для представлення нечіткої множини "невеликий", а потім вказати діапазон значень, що відповідає цьому поняттю, наприклад, від 0 до 0.3. Такий інтерфейс дозволяє експертам зручно виражати свої моделі висловлювань та використовувати їх у подальшому аналізі та роботі з нечіткими множинами.

Підпис та легенда в графічному виборі моделі висловлювання можуть бути корисними для зручності сприйняття та інтерпретації створених моделей експертами. Підписи дозволяють користувачам надати назву своїй моделі або конкретним точкам на графіку, щоб ідентифікувати їх. Наприклад, експерт може позначити точку на графіку як "мала кількість" або "висока інтенсивність".

Легенда може бути корисною для пояснення значень та символів, які використовуються в графіку. Вона може включати пояснення кожного графічного символу або малюнка, які представляють різні моделі

висловлювань. Наприклад, у легенді можуть бути перераховані різні форми (ціль або обмеження), кожна з яких може відповідати різним типам нечітких множин.

Застосування підписів та легенд дозволяє експертам чітко ідентифікувати свої моделі та розуміти їх значення при подальшому аналізі або співпраці з іншими експертами.

Функція перегляду та редагування дозволяє користувачам активно взаємодіяти зі своїми моделями висловлювань у зручний спосіб. Користувач може переглядати існуючі моделі для отримання загального уявлення про їхній вигляд та дані, які вони відображають. Під час редагування користувач може додавати, видаляти або переміщувати точки на графіку, що визначають форму нечіткої множини. Також вони можуть змінювати параметри моделі, такі як форма або розподіл, для досягнення бажаного результату. Перед збереженням змін користувач може переглянути попередній вигляд моделі, щоб впевнитися, що вони відображають їхні наміри та потреби. Зберігши зміни, користувач може легко відновити їх у майбутньому або продовжити редагувати за необхідності. Ці функції дозволяють користувачам адаптувати моделі до змінних умов та виражати свої унікальні підходи та уявлення без зайвих ускладнень.

Функціонал збереження та обміну даними в проекті полягає в забезпеченні зручного та ефективного способу зберігання та обміну всією інформацією, що стосується нечітких моделей висловлювань.

Ця функція включає в себе можливість зберігання всіх введених користувачем даних проекту, таких як параметри моделей, визначені експертами, результати аналізу, введені користувачем висновки та рекомендації.

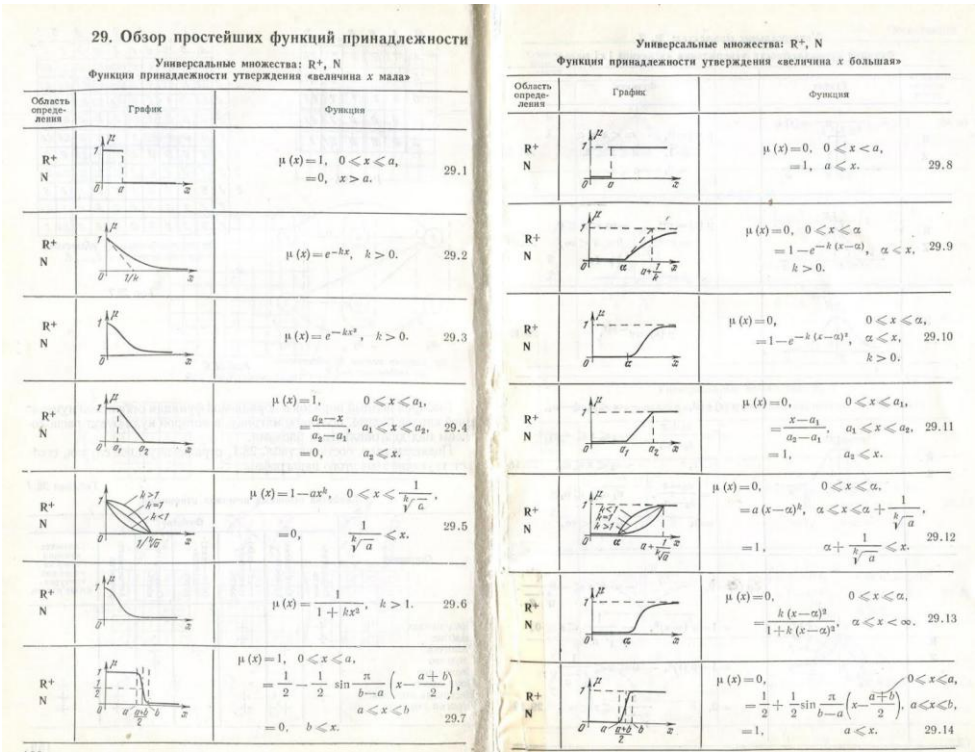
Крім того, система повинна забезпечувати зручний доступ до збереженої інформації та можливість обміну нею між учасниками проекту. Це дає змогу

експертам спільно працювати над проектом, обмінюючись думками та даними, а також дозволяє замовникам отримувати актуальну інформацію щодо ходу робіт та результатів аналізу. Такий обмін інформацією сприяє досягненню кращих результатів та підвищенню якості проекту в цілому.

Функція виведення результатів обчислень включає в себе представлення усіх обчислених даних у зручному та легкозрозумілому форматі для користувача.

У результаті обчислень всі дані, такі як параметри моделей, значення критеріїв або вагові коефіцієнти, будуть виводитися в окремому рядку таблиці. Це дозволяє користувачеві легко переглянути та аналізувати результати виконаних обчислень.

Також, функція також включає в себе можливість виведення графіків, що демонструють моделі висловлювань (приклади формул представлені в рис.2.1.2-2.1.4). Графіки допомагають візуалізувати результати та розуміти взаємозв'язки між різними параметрами моделі. Наприклад, графік може показати форму нечіткої функції належності або залежність вагових коефіцієнтів від введених експертами даних.



29. Обзор простейших функций принадлежности

Универсальные множества: R+, N

Функция принадлежности утверждения «величина x мала»

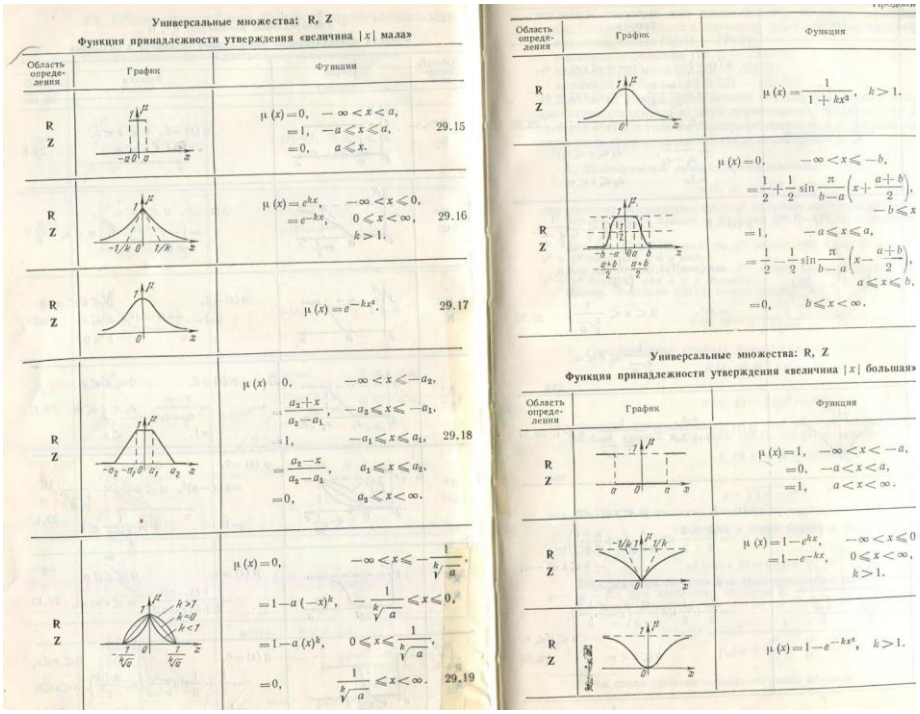
Область определения	График	Функция
R+ N		$\mu(x)=1, \quad 0 \leqslant x \leqslant a,$ $=0, \quad x>a.$ 29.1
R+ N		$\mu(x)=e^{-kx}, \quad k>0.$ 29.2
R+ N		$\mu(x)=e^{-kx^2}, \quad k>0.$ 29.3
R+ N		$\mu(x)=1, \quad 0 \leqslant x \leqslant a_1,$ $=\frac{a_2-x}{a_2-a_1}, \quad a_1 \leqslant x \leqslant a_2,$ $=0, \quad a_2 \leqslant x.$ 29.4
R+ N		$\mu(x)=1-ax^k, \quad 0 \leqslant x \leqslant \frac{1}{\sqrt[k]{a}},$ $=0, \quad \frac{1}{\sqrt[k]{a}} \leqslant x.$ 29.5
R+ N		$\mu(x)=\frac{1}{1+kx^2}, \quad k>1.$ 29.6
R+ N		$\mu(x)=1, \quad 0 \leqslant x \leqslant a,$ $=\frac{1}{2}-\frac{1}{2}\sin\frac{\pi}{b-a}\left(x-\frac{a+b}{2}\right), \quad a<x<b$ $=0, \quad b \leqslant x.$ 29.7

Универсальные множества: R+, N

Функция принадлежности утверждения «величина x большая»

Область определения	График	Функция
R+ N		$\mu(x)=0, \quad 0 \leqslant x < a,$ $=1, \quad a \leqslant x.$ 29.8
R+ N		$\mu(x)=0, \quad 0 \leqslant x \leqslant \alpha$ $=1-e^{-k(x-\alpha)}, \quad \alpha < x,$ $k>0.$ 29.9
R+ N		$\mu(x)=0, \quad 0 \leqslant x \leqslant \alpha,$ $=1-e^{-k(x-\alpha)^2}, \quad \alpha < x,$ $k>0.$ 29.10
R+ N		$\mu(x)=0, \quad 0 \leqslant x \leqslant a_1,$ $=\frac{x-a_1}{a_2-a_1}, \quad a_1 \leqslant x \leqslant a_2,$ $=1, \quad a_2 \leqslant x.$ 29.11
R+ N		$\mu(x)=0, \quad 0 \leqslant x \leqslant \alpha,$ $=a(x-\alpha)^k, \quad \alpha \leqslant x \leqslant \alpha+\frac{1}{\sqrt[k]{a}},$ $=1, \quad \alpha+\frac{1}{\sqrt[k]{a}} \leqslant x.$ 29.12
R+ N		$\mu(x)=0, \quad 0 \leqslant x \leqslant \alpha,$ $=\frac{k(x-\alpha)^2}{1+k(x-\alpha)^2}, \quad \alpha \leqslant x < \infty.$ 29.13
R+ N		$\mu(x)=0, \quad 0 \leqslant x \leqslant a,$ $=\frac{1}{2}+\frac{1}{2}\sin\frac{\pi}{b-a}\left(x-\frac{a+b}{2}\right), \quad a \leqslant x \leqslant b,$ $=1, \quad a \leqslant x.$ 29.14

(рис2.1.2)



Универсальные множества: R, Z

Функция принадлежности утверждения «величина $|x|$ мала»

Область определения	График	Функция
R Z		$\mu(x)=0, \quad -\infty<x<a,$ $=1, \quad -a\leqslant x\leqslant a,$ $=0, \quad a<x.$ 29.15
R Z		$\mu(x)=e^{kx}, \quad -\infty<x\leqslant 0,$ $=e^{-kx}, \quad 0\leqslant x<\infty,$ $k>1.$ 29.16
R Z		$\mu(x)=e^{-kx^2}.$ 29.17
R Z		$\mu(x)=0, \quad -\infty<x\leqslant -a_2,$ $=\frac{a_2+x}{a_2-a_1}, \quad -a_2\leqslant x\leqslant -a_1,$ $=1, \quad -a_1\leqslant x\leqslant a_1,$ 29.18 $=\frac{a_2-x}{a_2-a_1}, \quad a_1\leqslant x\leqslant a_2,$ $=0, \quad a_2\leqslant x<\infty.$
R Z		$\mu(x)=0, \quad -\infty<x\leqslant -\frac{1}{\sqrt[k]{a}},$ $=1-a(-x)^k, \quad -\frac{1}{\sqrt[k]{a}}\leqslant x\leqslant 0,$ $=1-a(x)^k, \quad 0\leqslant x\leqslant \frac{1}{\sqrt[k]{a}},$ $=0, \quad \frac{1}{\sqrt[k]{a}}\leqslant x<\infty.$ 29.19

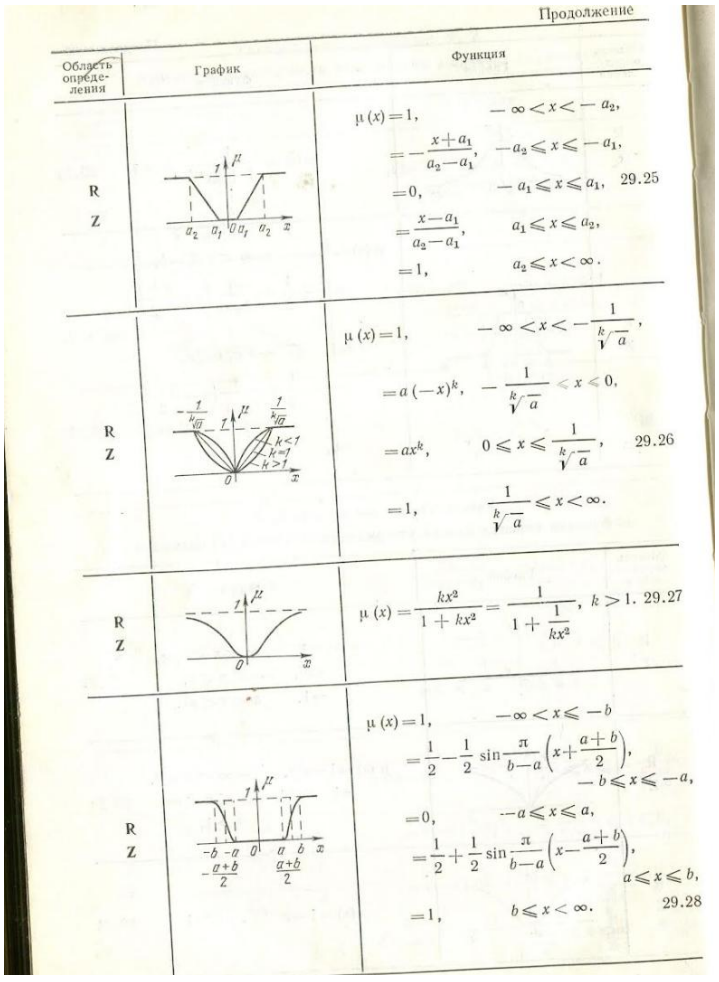
Область определения	График	Функция
R Z		$\mu(x)=\frac{1}{1+kx^2}, \quad k>1.$
R Z		$\mu(x)=0, \quad -\infty<x\leqslant -b,$ $=\frac{1}{2}+\frac{1}{2}\sin\frac{\pi}{b-a}\left(x+\frac{a+b}{2}\right), \quad -b\leqslant x$ $=1, \quad -a\leqslant x\leqslant a,$ $=\frac{1}{2}-\frac{1}{2}\sin\frac{\pi}{b-a}\left(x-\frac{a+b}{2}\right), \quad a\leqslant x\leqslant b,$ $=0, \quad b\leqslant x<\infty.$

Универсальные множества: R, Z

Функция принадлежности утверждения «величина $|x|$ большая»

Область определения	График	Функция
R Z		$\mu(x)=1, \quad -\infty<x<-a,$ $=0, \quad -a<x<a,$ $=1, \quad a<x<\infty.$
R Z		$\mu(x)=1-e^{kx}, \quad -\infty<x\leqslant 0$ $=1-e^{-kx}, \quad 0\leqslant x<\infty,$ $k>1.$
R Z		$\mu(x)=1-e^{-kx^2}, \quad k>1.$

(рис.2.1.3)



Продолжение

Область определения	График	Функция
R Z		$\mu(x)=1, \quad -\infty<x<-a_2,$ $=-\frac{x+a_1}{a_2-a_1}, \quad -a_2\leqslant x\leqslant -a_1,$ $=0, \quad -a_1\leqslant x\leqslant a_1,$ 29.25 $=\frac{x-a_1}{a_2-a_1}, \quad a_1\leqslant x\leqslant a_2,$ $=1, \quad a_2\leqslant x<\infty.$
R Z		$\mu(x)=1, \quad -\infty<x<-\frac{1}{\sqrt[k]{a}},$ $=a(-x)^k, \quad -\frac{1}{\sqrt[k]{a}}<x\leqslant 0,$ $=ax^k, \quad 0\leqslant x\leqslant \frac{1}{\sqrt[k]{a}},$ 29.26 $=1, \quad \frac{1}{\sqrt[k]{a}}\leqslant x<\infty.$
R Z		$\mu(x)=\frac{kx^2}{1+kx^2}=\frac{1}{1+\frac{1}{kx^2}}, \quad k>1.$ 29.27
R Z		$\mu(x)=1, \quad -\infty<x\leqslant -b$ $=\frac{1}{2}-\frac{1}{2}\sin\frac{\pi}{b-a}\left(x+\frac{a+b}{2}\right), \quad -b\leqslant x\leqslant -a,$ $=0, \quad -a\leqslant x\leqslant a,$ $=\frac{1}{2}+\frac{1}{2}\sin\frac{\pi}{b-a}\left(x-\frac{a+b}{2}\right), \quad a\leqslant x\leqslant b,$ $=1, \quad b\leqslant x<\infty.$ 29.28

(рис.2.1.4)

Такий підхід до виведення результатів обчислень дозволяє користувачам отримати повний обсяг інформації про процес аналізу та результати його виконання у зручному та зрозумілому форматі.

2.2. Розробка алгоритму обробки графічних даних.

Для розробки алгоритму обробки графічних даних, що представлені у вигляді нечітких множин, потрібно спроектувати процес перетворення цих графічних представлень в числові дані, що знаходяться в межах від 0 до 1.

Алгоритм має передбачати перетворення графічних образів нечітких множин на числові значення, які відображають ступінь належності кожної точки до цієї множини. Це може включати в себе аналіз форми та розмірів графічних об'єктів, їх положення на площині, та інші параметри, що визначають характеристики нечіткої множини.

Далі алгоритм повинен обчислити числові значення для кожної точки, враховуючи їхнє розташування та характеристики в графічному представленні. Ці значення повинні відповідати ступеню належності кожної точки до нечіткої множини, де 0 відповідає повному відсутності належності, а 1 - повному належності.

Остаточним етапом буде перевірка алгоритму на різноманітних графічних образах нечітких множин і коригування параметрів алгоритму для досягнення оптимальної точності та ефективності перетворення графічних даних в числові значення.

(рис.2.2.1)

У цій діаграмі ми маємо три компоненти:

- Вхідні графічні дані: це вихідні дані у формі графічних об'єктів, що представляють нечіткі множини.
- Алгоритм обробки: цей компонент відповідає за обробку вхідних графічних даних та їх перетворення у числові значення в межах від 0 до 1.
- Числові дані: це результат обробки, який представляє собою числові дані, які відображають ступінь належності кожної точки до відповідної нечіткої множини.

Ця діаграма (рис2.2.1) демонструє основні кроки процесу графічної обробки нечітких множин у програмі.

2.3. Впровадження формули Шортліфа для розрахунку оцінки якості проекту

Впровадження формули Шортліфа для розрахунку оцінки якості проекту є ключовим етапом в розробці програмного застосунку, оскільки ця формула надає структурований метод оцінки проектів на основі числових даних. Цей етап передбачає перетворення теоретичної моделі в конкретну реалізацію в програмному коді.

Під час імплементації формули Шортліфа розробник повинен створити функцію або метод, який отримує на вхід необхідні числові параметри, такі як вагомість кожного критерію, їх значення та важливість для проекту. Потім ця функція виконує необхідні математичні операції, описані у формулі Шортліфа(рис.2.3.1), для отримання підсумкового числового результату - оцінки якості проекту.

$$\text{КУ}[h: e_1, e_2] = \text{МД}[h: e_1] + \text{МН}[h: e_2](1 - \text{МД}[h: e_1])$$

(рис.2.3.1)

У процесі імплементації слід враховувати особливості формули Шортліфа, такі як вагомість кожного критерію, методи нормалізації даних та обчислення їх суми. Також важливо врахувати можливі варіанти адаптації формули під конкретні потреби проекту, що може включати в себе розширення функціоналу чи введення додаткових параметрів.

Імплементація формули Шортліфа дозволяє автоматизувати процес оцінки якості проекту, забезпечуючи об'єктивність та точність результатів. Після успішної реалізації цього етапу програмний застосунок буде здатний розраховувати оцінку якості проекту на основі введених числових даних, що використовуються для подальшого аналізу, виведення користувачу або збереження у базі даних.

Висновки

Аналіз результатів розробленого програмного застосунку, який використовує теорію нечітких множин для оцінки якості проектів на основі статичних даних, дозволяє зробити наступні висновки:

- Ефективність використання теорії нечітких множин: Використання теорії нечітких множин у програмному застосунку дозволило експертам представляти та обробляти нечіткі дані у зручний спосіб. Це сприяло покращенню об'єктивності оцінки якості проектів, оскільки дозволило враховувати неоднозначність та нечіткість вхідних даних.

- Точність результатів: Результати, отримані за допомогою розробленого програмного застосунку, відображають ступінь якості проектів на основі введених статичних даних. Точність цих результатів залежить від якості введених даних та коректності використання теорії нечітких множин у процесі оцінки.

- Зручність та ефективність інструментів аналізу: Програмний застосунок має зручний інтерфейс, що дозволяє експертам використовувати різноманітні інструменти аналізу та візуалізації даних. Це сприяє швидкому та зручному проведенню оцінки якості проектів та робить процес більш ефективним.

- Можливості аналізу та управління даними: Програмний застосунок забезпечує можливості аналізу та управління введеними даними, що дозволяє експертам проводити детальний аналіз та коригувати параметри оцінки для досягнення кращих результатів.

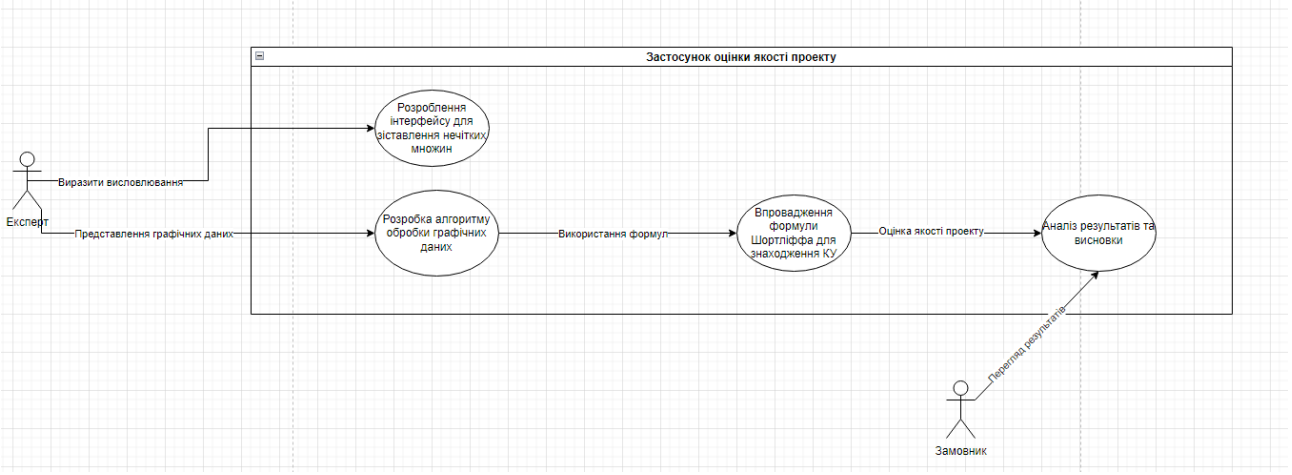
Загалом, аналіз результатів демонструє, що використання теорії нечітких множин у програмному застосунку для оцінки якості проектів на основі статичних даних є ефективним і дозволяє отримувати об'єктивні та точні результати. Однак успішність такого підходу залежить від якості введених даних та правильного використання інструментів аналізу.

РОЗДІЛ 3

РЕАЛІЗАЦІЯ РОЗРОБКИ ЗАСТОСУНКУ

3.1. Архітектура застосунку

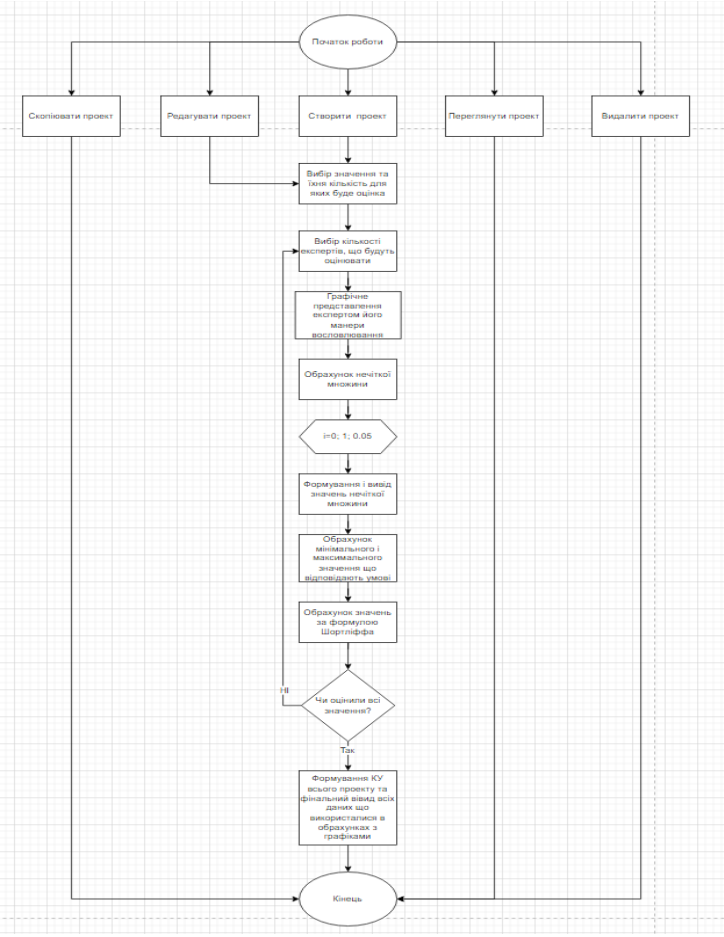
Даний програмний застосунок буде максимально простий, для того, щоб експерти могли більш легше його зрозуміти, буде мінімалізовано кількість кнопок, щоб побудова оцінки якості проекту не займала багато часу, а представлення графічної моделі, експерти могли ввести вручну, що дасть змогу більш точніше висловити свою точку бачення даного модулю, або ж якщо експерт буде присутній на оцінці всіх модулів, то оцінки всього проекту в цілому.



(рис3.1.1)

Наведена (рис 3.1.1) діаграма прецедентів показує відношення між експертом і замовником, та окремі модулі, що будуть задіяні в програмі, на які будуть скидатися основні зусилля, та будуть мати централізований спектр уваги, в розробці програмного застосунку.

Щоб довести в зручному форматі для розуміння, логіку програми та спектр розробки, я створив діаграму програми де графічно відображається логіка програми, яка буде задіяна в ПЗ для обрахунку оцінки якості певного проекту на основі статичних даних, і графічному представленню своєї логіки певним кругом експертів, що будуть прийняті для висловлювання своєї думки. Даний графік зображений на малюнку 3.1.2.



(рис3.1.2)

3.2. Вибір мови програмування

В процесі розробки програмного засобу для оцінки якості проектів на основі теорії нечітких множин та формули Шортлліфа, було використано ряд бібліотек та інструментів, які допомагають ефективно обробляти дані, візуалізувати результати та створювати користувацький інтерфейс. Ось детальний опис використаних бібліотек та їх функціоналу в даному програмному засобі:

- *NumPy (np): NumPy* - це бібліотека для мови програмування *Python*, яка додає підтримку для великих масивів та матриць, разом з великою колекцією математичних функцій, що дозволяє ефективно виконувати операції з числовими даними. У даній програмі *NumPy* використовується для обчислення результатів за формулою Шортліфа та для маніпулювання числовими даними.
- *Pandas (pd): Pandas* - це бібліотека для обробки та аналізу даних у *Python*. У програмі вона використовується для завантаження та обробки даних проектів, збережених у базі даних або у файловій системі, а також для побудови таблиць та аналізу даних.

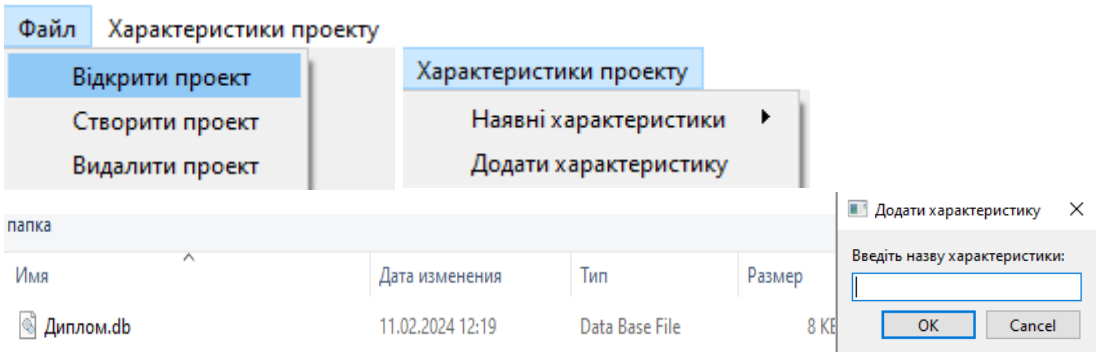
- *PyQtGraph (pg): PyQtGraph* - це бібліотека для візуалізації наукових даних та графічного відображення результатів. Вона використовується для створення графіків та візуалізації нечітких множин, що дозволяє користувачеві краще розуміти дані та результати оцінки проектів.
- *Matplotlib.pyplot (plt): Matplotlib* - це бібліотека для візуалізації даних у *Python*. У програмі вона використовується для побудови графіків та діаграм, що представляють результати оцінки якості проектів та інші аспекти аналізу даних.
- *PyQt6: PyQt6* - це набір *Python*-зв'язків для бібліотеки *Qt6*. Використовується для створення графічного інтерфейсу користувача (*GUI*) програми, включаючи кнопки, таблиці, вікна та інші елементи інтерфейсу.
- *SQLite3: SQLite3* - це вбудована СУБД, яка дозволяє зберігати та обробляти дані у локальній базі даних. У програмі використовується для зберігання та керування даними проектів.

Ці бібліотеки та інструменти використовуються в поєднанні один з одним для реалізації функціоналу програмного засобу, який включає в себе завантаження та обробку даних проектів, обчислення оцінки якості проекту за допомогою формули Шортлліфа, візуалізацію результатів та створення користувацького інтерфейсу для зручного користування програмою.

3.3. Опис функціоналу програми

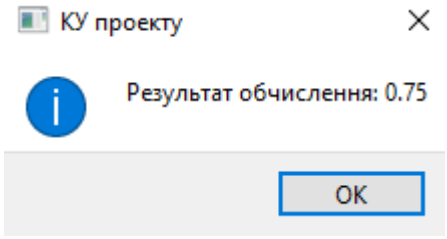
Цей застосунок є інтерфейсом для керування та аналізу проектів, який включає в себе ряд функцій для зручного взаємодії з користувачем. Основний вікно застосунку складається з різних елементів, таких як кнопки, меню та поля введення, які спрощують використання програми.

Головне вікно програми містить меню з можливістю відкриття, створення та видалення проектів. Користувач може також переглядати та редагувати характеристики проектів через це меню(рис.3.3.1).

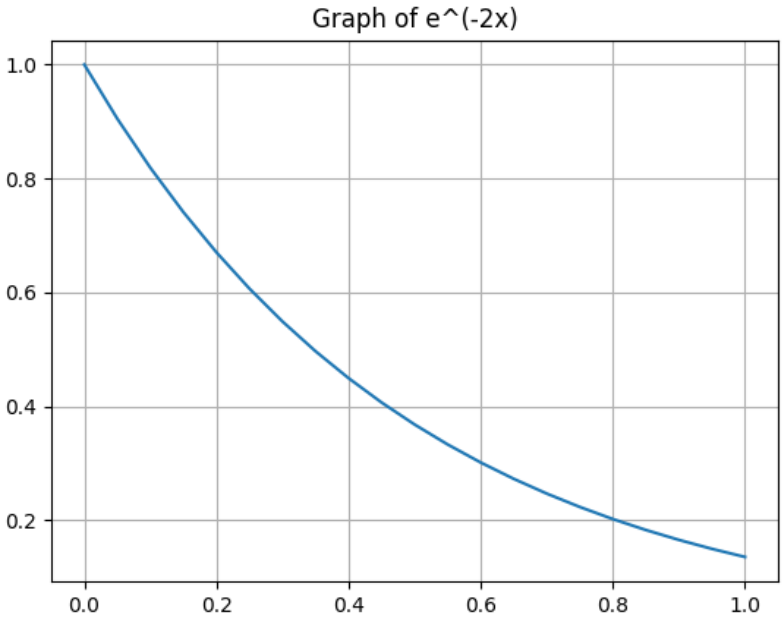


(рис3.3.1)

Окрім того, програма має кнопки для виконання певних дій, таких як додавання нових експертів, обчислення формули Шортлліфа (рис3.3.2) та побудова графіків(рис3.3.3). Ці кнопки знаходяться у верхній частині вікна і дозволяють користувачеві швидко виконувати потрібні дії.

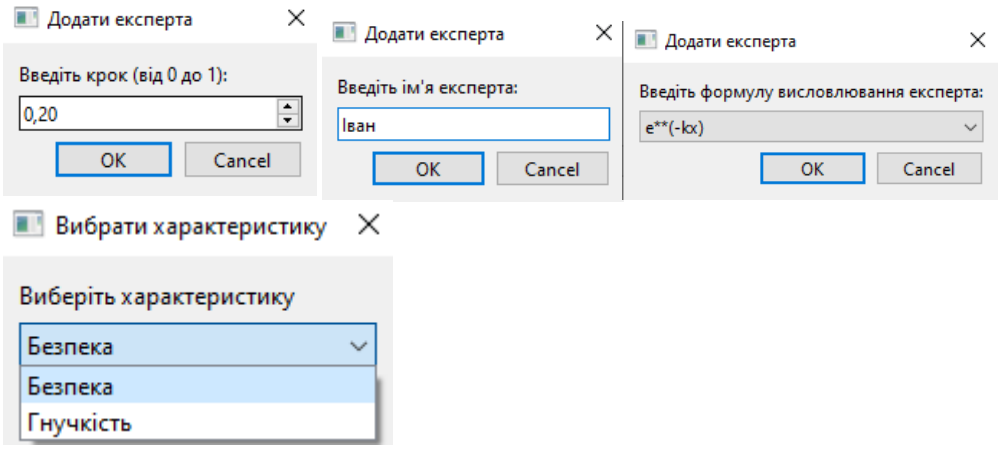


(рис3.3.2)



(рис.3.3.3)

Крім того, в програмі існують вікна для введення даних, таких як назва крок графічного представлення моделі висловлювання, ім’я експерта та вибір характеристики що буде оцінюватися. Ці вікна з'являються при виконанні певних дій користувачем і спрощують введення необхідної інформації (рис3.3.4).



(рис3.3.4)

Усі дані проекти зберігаються в базі даних *SQLite*, що дозволяє користувачеві зручно керувати та аналізувати їх. Користувач може відкривати та змінювати існуючі проекти або створювати нові, а також видаляти непотрібні. Це забезпечує зручну роботу з програмою та ефективне використання її функціоналу.

Цей застосунок розроблено з використанням мови програмування *Python* та ряду бібліотек, таких як *PyQt6*, *Matplotlib*, *NumPy*, *pandas* та *sqlite3*, що забезпечує його стабільну роботу та надійність.

Висновки

У розробленому застосунку використовується теорія нечітких множин для оцінки якості проектів. Це передбачає, що експерти, які мають різний досвід та підходи до оцінки проектів, можуть висловлювати свою думку щодо різних аспектів проекту, представляючи їх у вигляді нечітких множин. Замість того, щоб обмежуватися конкретним числовим значенням, вони вказують ступінь належності своїх оцінок до певної категорії або групи.

Після того, як експерти висловлять свої думки та визначать ступінь належності до різних аспектів проекту, ці дані обробляються за допомогою спеціальних методів та алгоритмів, які базуються на теорії нечітких множин. Це дозволяє враховувати неоднозначність та нечіткість в оцінках, які часто виявляються в реальних умовах.

Отримані значення ступеня належності для кожного експерта та кожної характеристики проекту потім зберігаються в базі даних. Ця інформація служить основою для подальшого обчислення загального показника оцінки якості проекту, з використанням формули Шортлліфа або іншого відповідного методу. Результати цього обчислення можуть допомогти приймати

обґрунтовані рішення з урахуванням думки кожного експерта та їхнього внеску в оцінку проекту.

Такий підхід дозволяє отримати більш об'єктивну та реалістичну оцінку якості проекту, оскільки враховує різноманітність точок зору та експертних думок. Крім того, збереження даних у базі даних дозволяє легко виконувати подальший аналіз та маніпулювання результатами для отримання більш глибокого розуміння проекту та його потенційних перспектив.

Висновок

В результаті дослідження та розробки застосунку для оцінки якості проектів на основі теорії нечітких множин та статичних даних було виявлено кілька ключових аспектів, які варто врахувати в контексті загального внеску та можливостей цієї роботи.

Дослідження підтвердило потужність та релевантність теорії нечітких множин в сучасному управлінні проектами. Застосування цієї теорії дозволяє ефективно враховувати невизначеність та неоднозначність в процесі прийняття рішень, що є необхідним для успішного керування проектами у складних умовах.

Розроблений застосунок відкриває широкі можливості для використання в практичній діяльності. Він може бути застосований в різних сферах, включаючи інженерію, інформаційні технології, будівництво, фінанси та інші, де необхідно оцінювати якість проектів з використанням статичних даних.

Дослідження виявило можливості для подальшого розвитку та удосконалення методів оцінки якості проектів. Зокрема, варто розглянути можливість використання більш складних моделей нечітких множин, а також розширення функціоналу застосунку для роботи з динамічними даними та врахування додаткових факторів в оцінці.

В цілому, дана робота не лише досліджує і розробляє конкретний застосунок, а й вносить важливий внесок у загальне розуміння та розвиток методів управління проектами. Вона сприяє вдосконаленню інструментарію та підходів у сфері управління проектами, що в свою чергу сприятиме покращенню ефективності та результативності проектної діяльності в різних галузях та контекстах.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Лейко С. Г. Основи теорії нечітких множин : Навч. посіб. Одеса : Астропринт, 2005. 192 с.

2. Бьюкенен, Б. Г. та Шортліфф, Е. Г. (ред.). Експертні системи на основі правил: експерименти MYCIN Стенфордського проекту евристичного програмування. Reading, MA: Addison-Wesley, 1984.

3. Руденко Ю. МЕТОДИ НАВЧАННЯ ТЕОРІЇ НЕЧІТКИХ МНОЖИН СТУДЕНТІВ. *European Science*. 2020. Sge17-02. С. 53–64. URL: https://doi.org/10.30890/2709-2313.2023-17-02-028 (дата звернення: 22.01.2024).

4. ДСТУ ISO 10006:2018 Управління якістю. Настанови щодо управління якістю в проектах (ISO 10006:2017, IDT) − Київ: Держстандарт України, 2018. Чинний.

5. SQLite / K. P. Gaffney et al. *Proceedings of the VLDB Endowment*. 2022. Vol. 15, no. 12. P. 3535–3547. URL: https://doi.org/10.14778/3554821.3554842 (date of access: 27.01.2024).

6. Matplolib[Електроний ресурс] – режим доступу до ресурсу: https://w3schools.com/python/matplotlib_subplot.asp (дата звернення: 27.01.2024).

7. Willman J. M. Beginning Pyqt: A Hands-On Approach to GUI Programming with PyQt6. Apress L. P., 2022.