Міністерство освіти і науки України

Український державний університет науки і технологій

Факультет Комп'ютерні технології та системи
Кафедра Комп'ютерні інформаційні технології

Пояснювальна записка

до кваліфікаційної роботи
магістра

на тему: **«Дослідження патернів прив`язки даних за зчепленням»**
за освітньою програмою **12 Інженерія програмного забезпечення**
зі спеціальності: **121 Інженерія програмного забезпечення**

Виконав: студент групи «ПЗ2222» _____ /Владислав БАЖИН/

Керівник: _____ /доц. Олена КУРОП'ЯТНИК/

Нормоконтролер: _____ /Світлана ВОЛКОВА/

Засвідчую, що у цій роботі немає запозичень з праць інших авторів без відповідних посилань
Студент _____
(підпис)

Дніпро – 2024 рік

Ministry of Education and Science of Ukraine

Ukrainian State University of Science and Technologies

Faculty Computer technologies and systems
Department Computer information technology

Explanatory Note

to Master's Thesis

on the topic: «Study of data binding patterns by coupling»

according to educational curriculum **12 software engineering**
in the Speciality: **121 software engineering**

Done by the student of the group PZ2222: /Vladyslav BAZHYN/

Scientific Supervisor: /Olena KUROPIATNYK/

Normative controller: /Svitlana VOLKOVA/

Dnipro – 2024

Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет: Комп'ютерних технологій і систем
Кафедра: Комп’ютерні інформаційні технології
Рівень вищої освіти: магістр
Освітня програма: Інженерія програмного забезпечення
Спеціальність: Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ____КІТ
_____ Вадим ГОРЯЧКІН
_____ січень 2023 р.

ЗАВДАННЯ

На кваліфікаційну роботу ОС Магістр

студенту Бажину Владиславу Андрійовичу

1. Тема дипломної роботи:
   «Дослідження патернів прив’язки даних за зчепленням»
   Керівник роботи: Куроп’ятник Олена Сергіївна
   затверджені наказом 1196 ст. від 05.12. 2022 року
2. Строк подання студентом роботи 12.01. 2024 року

3. Вихідні дані до дипломної роботи:
   _____.
4. Зміст пояснювальної записки (перелік питань до розробки):
   4.1. Основна частина: Розділ 1 Аналіз сучасного стану дослідження патернів, Розділ 2 Обґрунтування методу використання теорії графів для визначення ефективності патернів, Розділ 3 Проєктування й розробка інструментального забезпечення для визначення ефективності патернів

4.2. прив’язки даних, Розділ 4 Дослідження ефективності патернів прив’язки даних;

5. Перелік демонстраційного матеріалу:
   5.1. презентація;
   5.2. демонстраційне відео роботи програми.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Постановка задачі, збір вимог до програмного забезпечення	25.01.23	
2	Огляд основних аналогів	10.02.23-12.02.23	
3	Розробка програми	01.05.23-30.06.23	
4	Тестування та налагодження програми	01.07.23-10.07.23	
5	Написання основного змісту пояснювальної записки	15.08.23-01.09.23	
6	Узгодження і затвердження програмної документації	15.09.23-16.09.23	
7	Подання кваліфікаційної роботи до кафедри	07.01.24	100%
8	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії	21.01.24	

Студент _____ Владислав БАЖИН
(підпис) (Ім’я ПРІЗВИЩЕ)

Керівник роботи _____ доц. Олена КУРОП'ЯТНИК
(підпис) (Ім’я ПРІЗВИЩЕ)

РЕФЕРАТ

Об’єктом дослідження є процес побудови архітектури багатомодульних систем з використанням патернів прив’язки даних.

Метою дослідження є розробка методу та засобів визначення ефективності використання патернів прив’язки даних.

Методи дослідження. Для вирішення поставлених задач було використано елементи теорії графів та методи лексичного аналізу текстів програм. Застосовано засоби та методи C#, .NET та WindowsForms.

Наукова новизна. Розроблено метод визначення ефективності використання патернів прив’язки даних на основі лексичного аналізу програмного коду.

Пояснювальна записка складається зі вступу, 4 розділів, висновків, бібліографічного списку та додатків.

Вступ описує суть, мету та актуальність роботи (2 сторінки).

Перший розділ: розкриття проблеми, огляд аналогів та визначення головних інструментів (10 сторінок).

Другий розділ: обґрунтування напрямку дослідження, розробка методу визначення ефективності патернів прив’язки даних (10 сторінок).

Третій розділ: описано процес проєктування та розробки ПЗ (13 сторінок).

Четвертий розділ: експериментальні дослідження ефективності використання патернів на основі розробленого методу (11 сторінок).

Додатки – технічне завдання, робочий проєкт, тези доповіді, проєкт наукової статті.

Таблиць – 18, рисунків – 31, бібліографія – 20 джерел.

Ключові слова: патерн, граф, прив’язка даних, ефективність патернів.

ЗМІСТ

Вступ 8

Розділ 1 Аналіз сучасного стану дослідження патернів 10

1.1 Призначення та сфера застосування 10

1.2 Аналіз існуючих програмних аналогів 11

1.3 Огляд літератури 18

1.4 Постановка задачі 19

Висновки до розділу 1 20

Розділ 2 Обґрунтування методу використання теорії графів для визначення ефективності патернів 21

2.1 Визначення основного напряму та мети дослідження 21

2.2 Розробка методу визначення ефективності патернів 21

Висновки до розділу 2 31

Розділ 3 Проєктування й розробка інструментального забезпечення для визначення ефективності патернів прив’язки даних 32

3.1 Зовнішнє проєктування 32

3.1.1 Вхідні дані 32

3.1.2 Вихідні дані 32

3.1.3 Функціональні вимоги 32

3.2 Базова архітектура системи 33

3.2.1 Сценарій роботи програми 33

3.2.2 Етапи методу 33

3.3 Внутрішнє проєктування 36

3.4 Розробка інтерфейсу користувача 39

Висновки до розділу 3 44

Розділ 4 Дослідження ефективності патернів прив’язки даних 45
4.1 Підготовка до експерименту 45
4.2 Проведення експерименту 51
4.3 Результати експерименту 54
Висновки до розділу 4 55
Висновки 57
Список літератури 58
Додатки 60

ВСТУП

Актуальність роботи. Дослідження патернів прив'язки даних за зчепленням залишається актуальним та важливим напрямком у галузі розробки програмного забезпечення. Прив'язка даних – це механізм, що дозволяє автоматично синхронізувати дані між інтерфейсом користувача і бізнес-логікою програми. Зі зростанням складності та вимог до сучасних додатків стає важливим ефективне управління даними та їх відображенням [8].

Нижче приведено кілька аспектів, що обґрунтовують важливість дослідження патернів прив'язки даних за зчепленням:

Мобільна та веб-розробка: з мобільними та веб-додатками, що мають складні інтерфейси та динамічні дані, прив'язка даних за зчепленням стає ключовим аспектом для забезпечення плавної та правильної роботи програми.

Modern application frameworks, так само як MVVM (Model-View-ViewModel), використовує дані ingestion як засоби відокремлення business models від представництва і практики тестування.

Розподілені системи: у розподілених системах, де дані можуть бути розкидані різними вузлами або сервісами, ефективна прив'язка даних відіграє важливу роль у забезпеченні цілісності та актуальності інформації.

Реактивне програмування: концепції реактивного програмування та потоків даних стають дедалі популярнішими, і прив'язка даних за зчепленням є важливою для реалізації реактивних систем, оскільки вона ґрунтується на потоках даних, а також є досить важливим вираження змін тощо.

Таким чином, дослідження патернів прив'язки даних за зчепленням являється актуальним та корисним напрямком, що сприяє більш ефективній та сучасній розробці програмного забезпечення.

Об’єктом дослідження є процес побудови архітектури багатомодульних систем з використанням патернів прив’язки даних.

Предметом дослідження є метод оцінки впливу патернів прив’язки даних на показники зчеплення.

Мета і завдання дослідження. Метою магістерської роботи є розробка методу та засобів визначення ефективності використання патернів прив’язки даних.

Для досягнення поставленої мети було вирішено такі задачі:

1) Дослідження програмних аналогів з використанням лексичного аналізу, або визначенням ефективності написаного коду;

2) Розробка методу визначення ефективності використання патернів прив’язки даних в окремих задачах;

3) Розробка програмного забезпечення для визначення ефективності, а також програм, що будуть досліджуватися.

Методи дослідження. Для вирішення поставлених задач було використано елементи теорії графів та методи лексичного аналізу текстів програм. Застосовано засоби та методи C#, .NET та WindowsForms.

Наукова новизна. Розроблено метод визначення ефективності використання патернів прив’язки даних на основі лексичного аналізу програмного коду.

Практичне значення. В подальшому результати дослідження можуть бути впроваджені у навчальному процесі при вивчені дисциплін, що передбачають проєктування програмних систем, за спеціальності 121 «Інженерія програмного забезпечення».

Апробація результатів дослідження та публікації. Під час виконання дослідницької роботи було опубліковано тези доповіді в збірці 84-ої Всеукраїнської науково-технічної конференції молодих учених, магістрантів та студентів «Наука і сталий розвиток транспорту 2023». Результати доповідалися на наукових семінарах кафедри Комп’ютерні інформаційні технології Українського державного університету науки та технологій від 05.10.23 та 09.11.23.

РОЗДІЛ 1 АНАЛІЗ СУЧАСНОГО СТАНУ ДОСЛІДЖЕННЯ ПАТЕРНІВ

1.1 Призначення та сфера застосування

Результатом даного дослідження буде метод, що дозволить визначати ефективність використання патернів прив'язки даних в окремих задачах. Патерни широко засовуються у сучасному програмуванні, особливо на високому рівні. Їх існує досить багато, тому відразу зорієнтуватися буває досить складно. Дана кваліфікаційна робота покладає початок детальному вивченню ефективності застосування тих, чи інших патернів та дозволяє й надалі розвивати цю тему. Розроблені програми у майбутньому можна буде використовувати під час навчання студентів технічних спеціальностей, пов'язаних з програмуванням тощо, а також для розробки більш значних аналогів програм, що дозволять визначати ефективність використання патернів автоматизовано для доцільнішого використання початковими програмістами.

Порівняння ефективності патернів може бути складним завданням, оскільки ефективність залежить від багатьох факторів, таких як контекст використання, розмір проєкту, потреби в збереженні ресурсів та інші фактори. Однак, існує кілька кроків, які можуть допомогти у цьому процесі:

1) визначити критерії ефективності: перш за все потрібно скласти список критеріїв, за яким можна оцінювати патерни. Це можуть бути такі фактори, як продуктивність, швидкодія, зрозумілість коду, масштабованість та інші. Кожен критерій повинен бути ясно визначений і вимірюваний.

2) зібрання даних: зібрати достатню кількість даних про кожен патерн, що перевіряється на ефективність: сфера використання, направленість, взаємодія з архітектурою тощо. Один із варіантів – оглянути наявні дослідження, публікації або статті про використання патернів.

3) аналіз і оцінка: оцінити кожен патерн за кожним критерієм. Використовуючи числові оцінки або шкали, які дозволяють порівнювати патерни. На-

приклад, можна використовувати оцінку від 1 до 5, де 1 – найнижчий рівень ефективності, а 5 – найвищий рівень ефективності.

4) важливість критеріїв: надати вагу кожному критерію відповідно до його значимості для поточного проєкту, або вимог. Деякі критерії можуть бути важливішими, ніж інші, тому варто врахувати це під час порівняння.

5) обчислення результатів: підрахувати сумарні бали для кожного патерна на основі їх оцінок та ваг критеріїв. Це допоможе отримати числове представлення ефективності кожного патерна.

6) контекст використання: ефективність патерна може залежати від конкретного контексту використання. Тобто, в одній програмі можуть бути ефективними два патерни, але в іншій лише один з них тощо.

Цей підхід може допомогти систематично порівняти ефективність патернів. Проте, визначення "найефективнішого" патерна може бути складним завданням, оскільки це залежить від багатьох факторів. Оцінка патернів є суб'єктивним процесом, і важливо враховувати індивідуальні потреби та контекст кожного проєкту.

1.2 Аналіз існуючих програмних аналогів

В Visual Studio (рис. 1.1) доступні розширені можливості для аналізу коду, які дозволяють виявляти проблеми з продуктивністю, швидкодією та використанням ресурсів. Основні функції аналізу в Visual Studio включають наступні:

Performance Profiler (профілювання продуктивності): Performance Profiler дозволяє вимірювати та аналізувати продуктивність коду розробника: зв’язки між класами, доречність використання змінних тощо. Розробник може використовувати його для виявлення «вузьких» місць, шукання повільно виконуючих методів або функцій, аналізу використання пам'яті та інших факторів, що впливають на продуктивність.

Memory Usage Profiler (профілювання використання пам'яті): ця функція допомагає виявити проблеми з використанням пам'яті в коді розробника. Вона надає звіти про використання пам'яті, виявляє витоки пам'яті,

неправильне управління пам'яттю та інші проблеми, які можуть призвести до проблем з продуктивністю або стабільністю програми. Детальніше про це можна прочитати на офіційному сайті learn.microsoft.com.

CPU Usage Profiler (профілювання використання процесора): ця функція дозволяє виміряти та аналізувати використання процесора кодом програміста. Розробник може використовувати її для виявлення методів або функцій, які займають багато часу процесора, та оптимізації їх для покращення продуктивності.

Energy Consumption Profiler (профілювання споживання енергії): ця функція дозволяє вимірювати споживання енергії вашою програмою. Вона надає інформацію про енергоефективність вашого коду, допомагає виявляти проблеми, що призводять до надмірного споживання енергії, та дає поради щодо оптимізації.

Ці функції профілювання можна запустити в Visual Studio з використанням Profiler-а (профілювальника), який забезпечує детальний аналіз коду та надає важливу інформацію для вдосконалення продуктивності та якості програмного забезпечення.

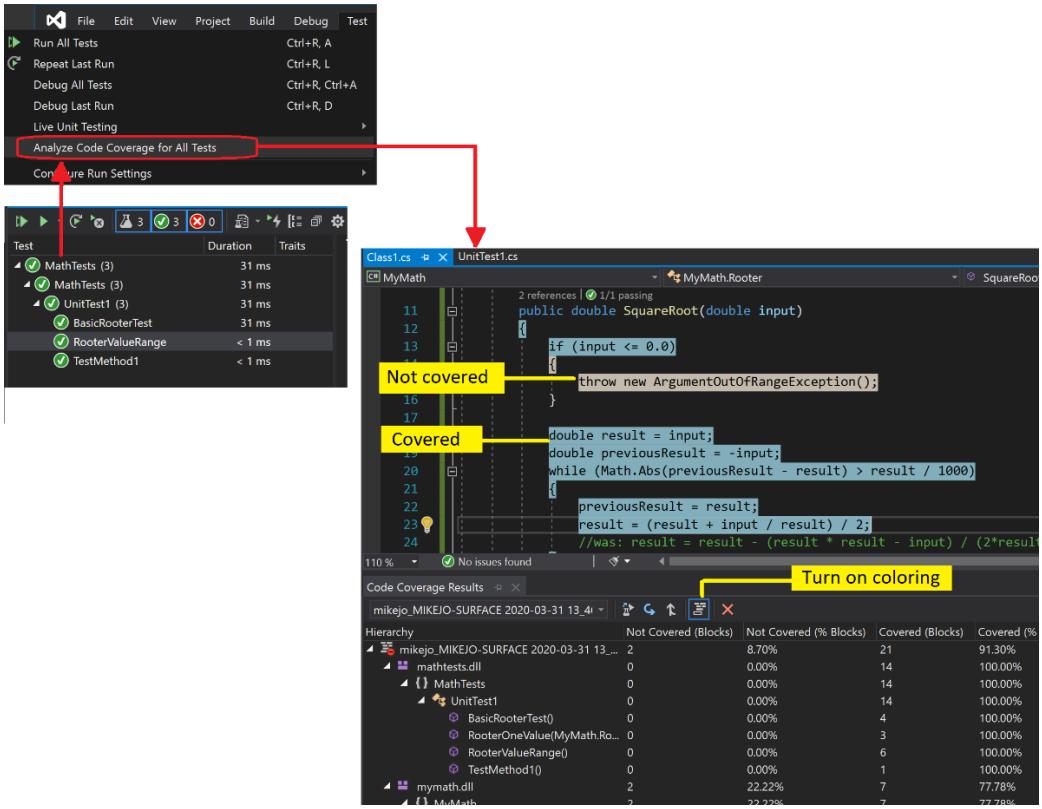

Рисунок 1.1 – Аналіз коду у Visual Studio

Аналіз коду в IntelliJ IDEA Community Edition 2020.2.1 x64 (рис. 1.2) включає різні функції та інструменти, які допомагають виявляти помилки, здійснювати перевірку якості коду, аналізувати його структуру та знаходити шляхи для його оптимізації. Ось кілька ключових функцій, які доступні в IntelliJ IDEA для аналізу коду:

- підказки та перевірки помилок: IntelliJ IDEA має вбудований розумний аналізатор, який надає підказки та виявляє потенційні помилки в коді. Розробник може побачити підказки про неправильне використання API, невизначені змінні, неправильну синтаксичну структуру та інші можливі проблеми;

- перейменування та переструктуризація: IntelliJ IDEA дозволяє легко перейменовувати змінні, функції, класи та інші елементи коду, автоматично оновлюючи всі посилання на них. Крім того, користувач може швидко переструктурувати код, переміщуючи його блоки, виділяючи методи, розділяючи класи тощо;

- пошук і заміна: IntelliJ IDEA має потужний пошуковий інструмент, який дозволяє знайти певний код або шаблон і замінити його іншим. Розробник може налаштувати різні параметри пошуку, такі як включення або виключення певних файлів чи каталогів, використання регулярних виразів тощо;
- вбудовані інструменти для тестування: IntelliJ IDEA надає можливості для автоматизованого тестування коду. Користувач може створювати і запускати тести одиниць, отримувати звіти про покриття коду тестами, виявляти проблеми в результатах тестів та шукати шляхи для їх виправлення;
- аналіз профілювання: IntelliJ IDEA має вбудовані інструменти для аналізу профілювання коду, які допомагають виявляти проблеми з продуктивністю та використанням пам'яті. Користувач може отримати детальну інформацію про час виконання різних частин коду, використання пам'яті та інші параметри;
- інтеграція з іншими інструментами: IntelliJ IDEA може інтегруватися з різними зовнішніми інструментами для аналізу коду, такими як SonarLint, Checkstyle, FindBugs тощо. Це дозволяє розробнику використовувати додаткові правила перевірки та виправлення помилок для покращення якості вашого коду.

Загалом, IntelliJ IDEA Community Edition 2020.2.1 x64 надає широкі можливості для аналізу та вдосконалення коду, допомагаючи зробити процес розробки більш продуктивним та ефективним.

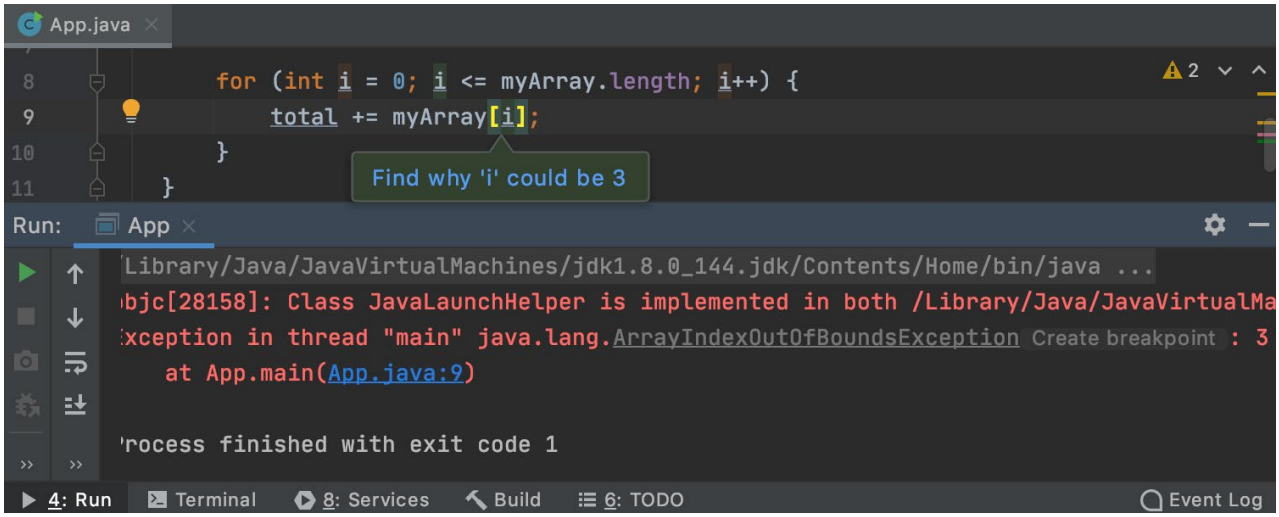

Рисунок 1.2 – Аналіз коду у IntelliJ IDEA

Graphviz – це програмне забезпечення з відкритим вихідним кодом для візуалізації графів та мереж. Воно надає інструменти створення діаграм, що представляють структурну інформацію як графів. Graphviz особливо корисний для аналізу та візуалізації складних структур та зв'язків у різних галузях, включаючи програмування, біоінформатику, соціальні мережі, та багато іншого. Головна сторінка (рис. 1.3) представляє собою ознайомчу інформацію, що допомагає користувачу розібратися з основним функціоналом.

Графи Graphviz зазвичай описуються з використанням DOT - простої текстової мови. У мові DOT визначаються вузли, ребра та його атрибути. Це дозволяє користувачам легко створювати та змінювати графічні структури. Робочу сторінка містить декілька прикладів з можливістю створити власний граф (рис. 1.4).

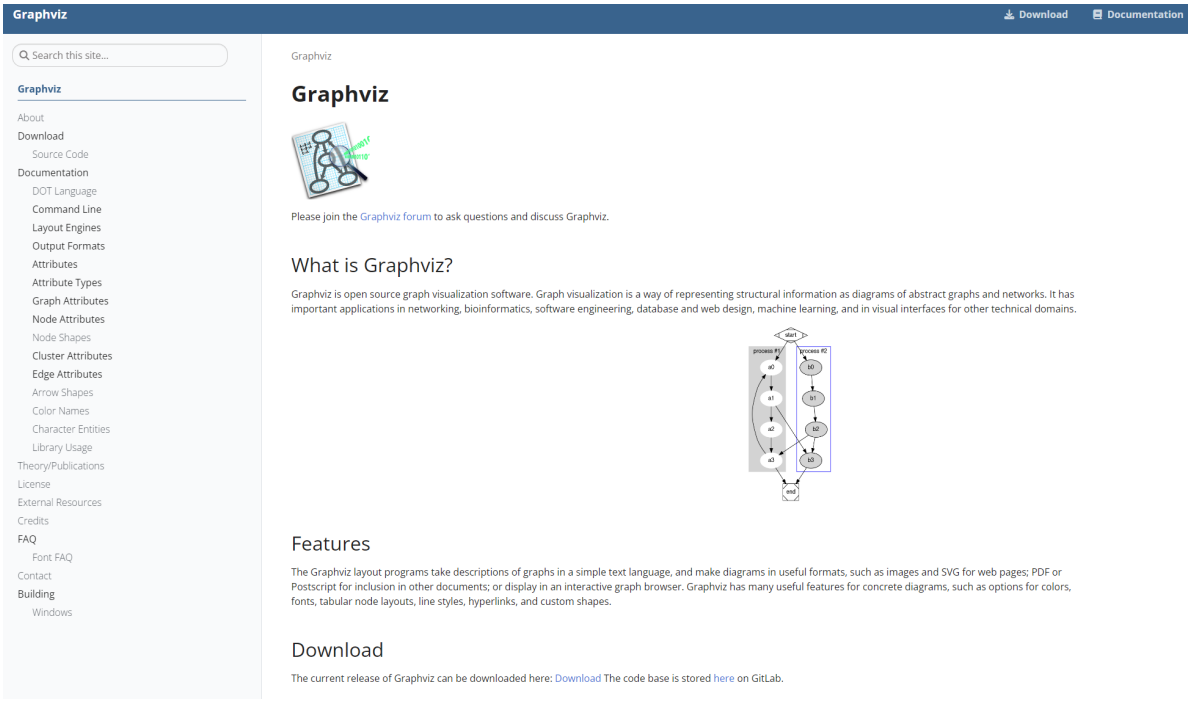

Рисунок 1.3 – Головна сторінка Graphviz

Graphviz підтримує різні алгоритми компонування, які визначають, як вузли та ребра мають бути розміщені на графі (рис. 1.5 та 1.6). Деякі з них включають DOT (ієрархічний розподіл), Neato (силова розкладка) та інші, кожен підходить для певних типів графів.

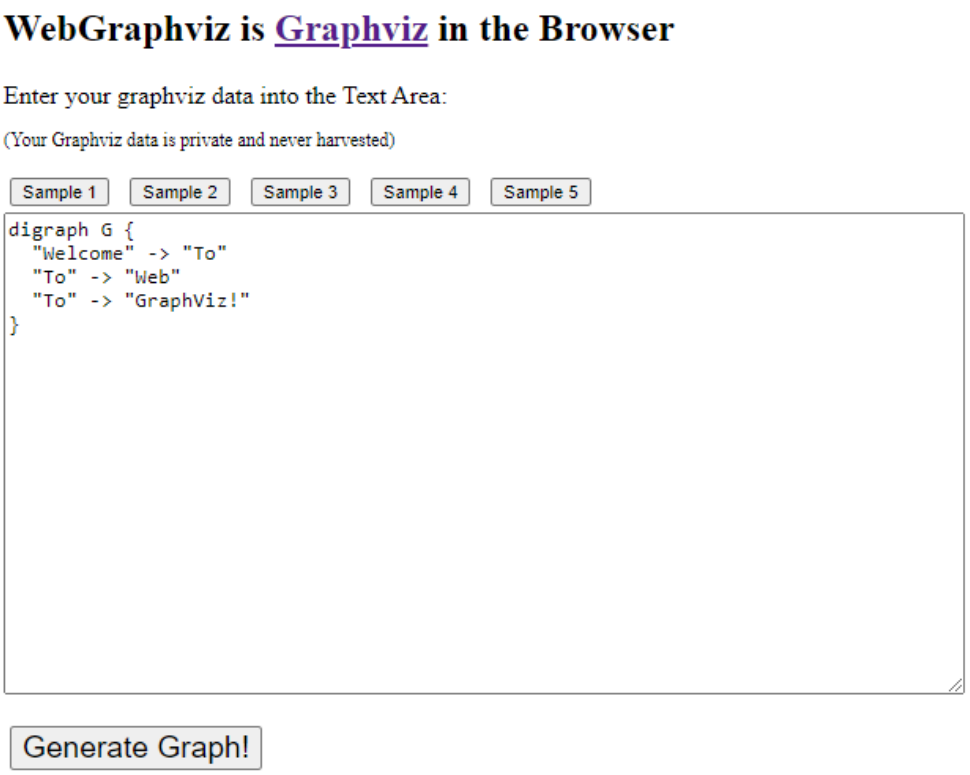

Рисунок 1.4 – Робоча сторінка Graphviz

Після визначення графа та вибору алгоритму компонування, Graphviz може візуалізувати граф у різних форматах, таких як PNG, PDF, SVG та інші. Це уможливлює створення інформативних діаграм.

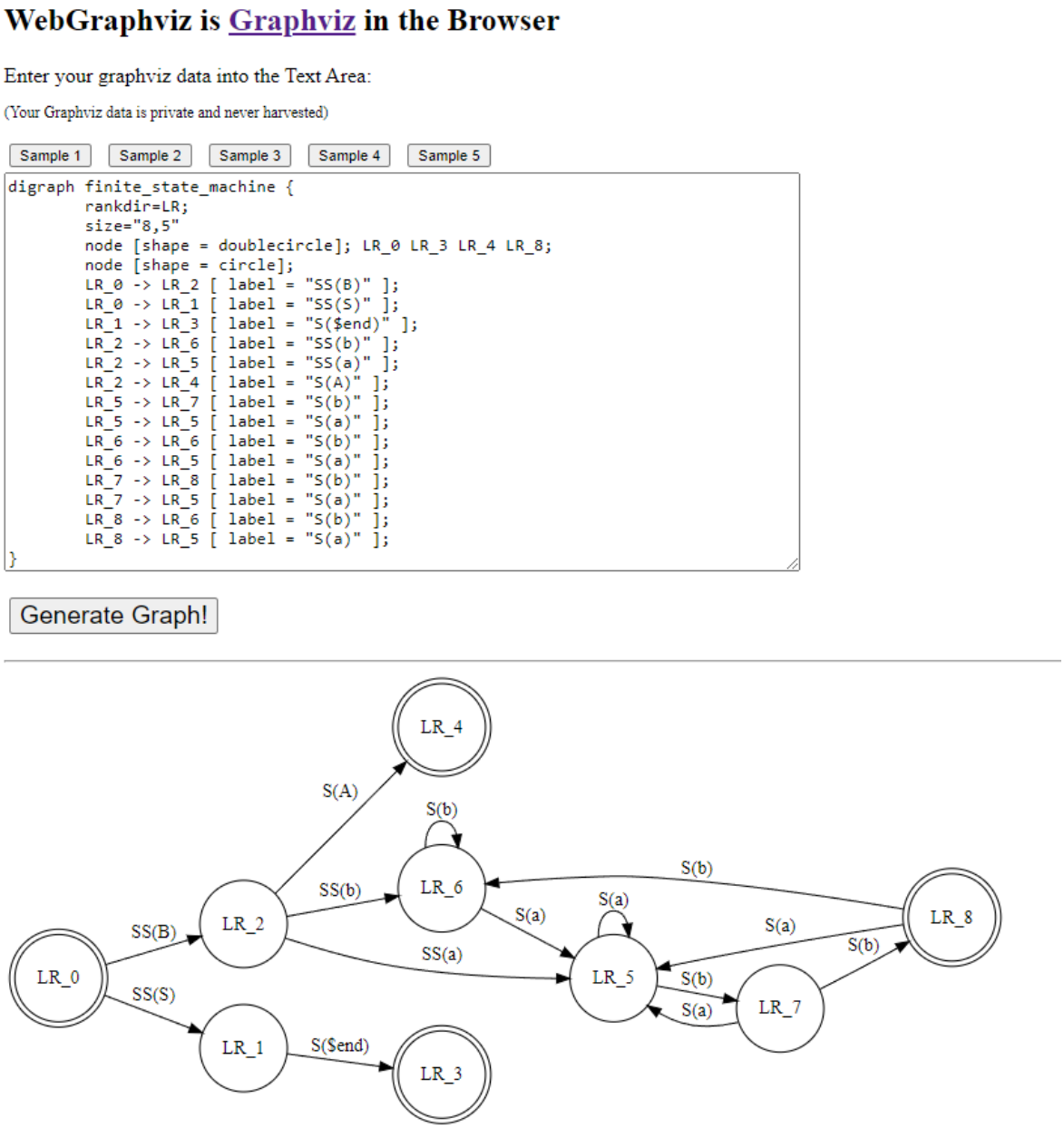

Рисунок 1.5 – Приклад графу у Graphviz

Graphviz підтримує різні формати виведення, що забезпечує гнучкість у використанні. Ви можете зберігати графи у різних форматах залежно від ваших потреб.

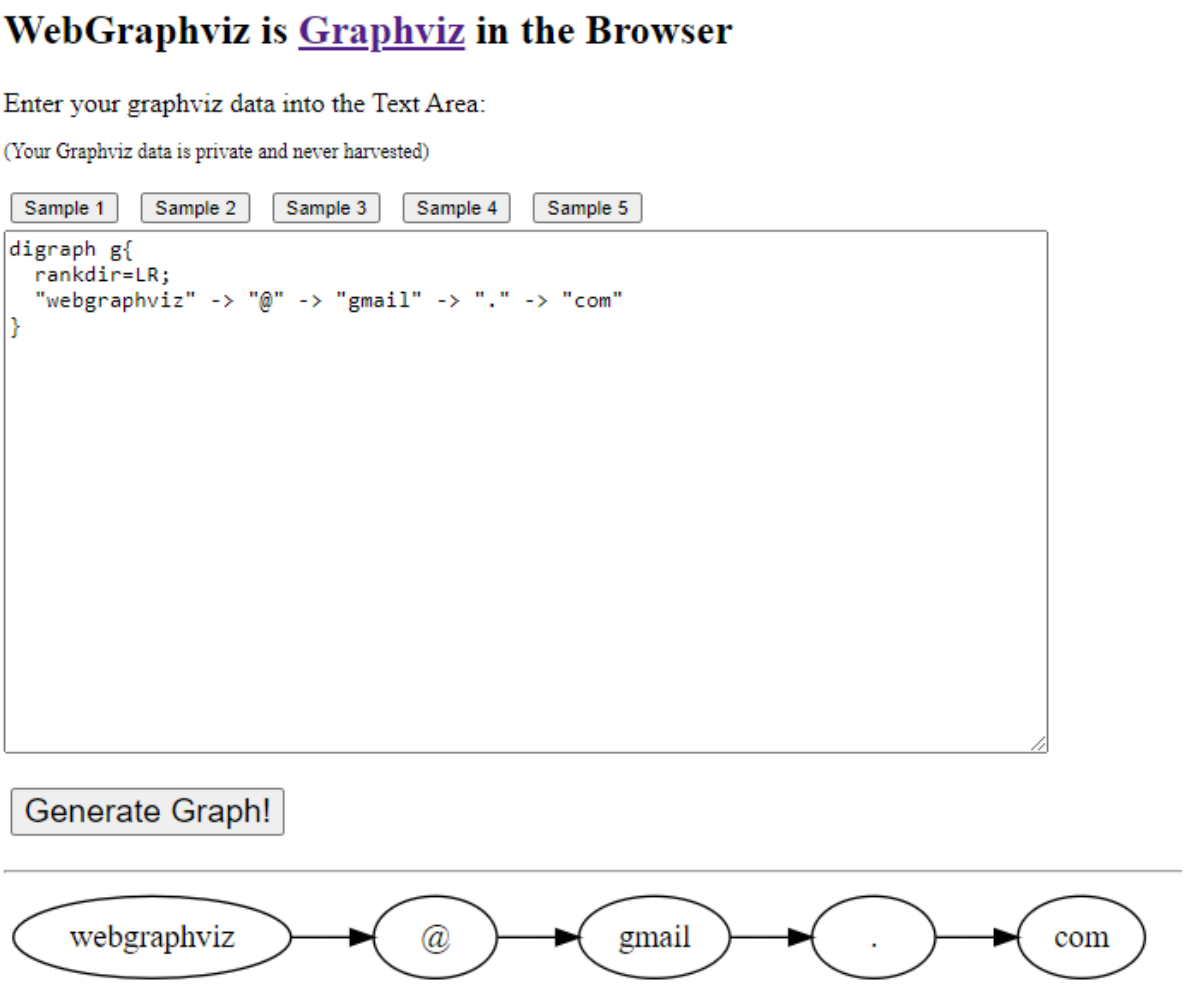

Рисунок 1.6 – приклад графу у Graphviz

Graphviz активно використовується в різних галузях, де важлива візуалізація структурних зв'язків, і він є потужним інструментом для аналізу та подання складних графових даних.

1.3 Огляд літератури

Згідно книги «Патерни проєктування на платформі .NET» [1], на платформі .NET для роботи з прив'язкою даних часто використовується принцип Model-View-ViewModel (MVVM). MVVM – це архітектурний патерн, який розділяє інтерфейс користувача на три основні компоненти:

1) Model (Модель): дані та бізнес-логіка програми. Це може бути клас, який надає дані для відображення, обробляє операції з даними та може повідомляти про свою зміну.

2) View (Подання): це – інтерфейс користувача, який відображає дані та обробляє введення користувача. View не містить бізнес-логіки; його завдання – просто відображати дані та забезпечувати взаємодію з користувачем.

3) ViewModel (Модель представлення): служить ланкою між Model і View. ViewModel містить логіку представлення, форматує дані для відображення у View та обробляє команди користувача. Він також може містити логіку взаємодії з Model і відстежувати зміни Model для оновлення даних в View.

Принцип використання патернів прив'язки даних на платформі .NET, зокрема, MVVM, включає наступні особливості:

- двостороння прив'язка (Two-Way Data Binding): Дозволяє автоматично синхронізувати дані між View та ViewModel. Якщо дані в ViewModel змінюються, вони автоматично оновлюються в View і навпаки;
- INotifyPropertyChanged: Інтерфейс INotifyPropertyChanged надає механізм сповіщень про зміну властивостей об'єкта. ViewModel реалізує цей інтерфейс, дозволяючи View підписуватись на повідомлення про зміну даних;
- Commands: Замість обробки подій безпосередньо з View, використовуються команди (Commands), які надають абстракцію для обробки дій користувача. Це допомагає зменшити зв'язаність і робить код тестованішим;
- Data Templates та Data Binding Markup Extensions: Використовуються для визначення того, як дані зв'язуються з елементами керування та як вони відображаються.

Застосування цих принципів спрощує розробку додатків, робить код структурованішим і підтримуваним, а також дозволяє створювати гнучкі і масштабовані інтерфейси на платформі .NET.

1.4 Постановка задачі

Для досягнення мети буде поставлено та вирішено такі задачі:

1) огляд та аналіз патернів, показників оцінки якості архітектури, її ефективності;

2) розробка критеріїв порівняння патернів;

3) розробка методу визначення ефективності використання патернів прив’язки даних на основі лексичного аналізу програмного коду.

4) написання програми, яка за цим методом буде оцінювати ефективність поданих на вхід програм з точки зору архітектури;

5) провести тестування методами білої та чорної скриньки та зробити налагодження програм;

6) дослідити вплив при виборі патернів на ефективність кінцевої програми й порівняти результатів між різними варіантами програм й патернів;

Висновки до розділу 1

У першому розділі було розглянуто призначення та сферу застосування дослідницької роботи. Оглянуто літературу, яка пов’язана з цією темою, що може допомогти у розв’язанні проблеми визначення ефективності патернів. Було розглянуто існуючі програмні аналоги, що мають схожий функціонал (наприклад, аналіз коду тощо). На основі цих пунктів поставлено задачу до подальшої дослідницької роботи.

РОЗДІЛ 2 ОБҐРУНТУВАННЯ МЕТОДУ ВИКОРИСТАННЯ ТЕОРІЇ ГРАФІВ ДЛЯ ВИЗНАЧЕННЯ ЕФЕКТИВНОСТІ ПАТЕРНІВ

2.1 Визначення основного напряму та мети дослідження

Метою дослідницької роботи є розробка методу та засобів визначення ефективності використання патернів прив’язки даних.

Основною проблемою під час виконання роботи є те, що немає конкретних засобів як можна точно визначити ефективність впливу патерну на програмний код. Тому, було виділено основні критерії, за якими буде проводитись це дослідження:

- кількість зв’язків між класами з використанням патерну та без нього;
- кількість рядків програмного коду в обох випадках;
- швидкодія програми в обох випадках;
- затрата апаратних ресурсів в обох випадках.

Під час перевірки будуть використовуватися два файли з програмним кодом. Це буде одна й та сама задача, яка буде вирішена з використанням патернів, а інша – без них. Після перевірки на основі отриманих даних буде сформовано висновок. Якщо кількість зв’язків з використанням патернів зменшиться, то результат буде позитивним. Кількість рядків програмного коду також повинна бути менша для позитивного результату. Швидкодія та затрата ресурсів у невеликих задачах не являються основним критерієм на сучасних комп’ютерах, але вони також будуть враховуватись, оскільки патерни прив’язки даних використовуються також у великих бізнес проєктах, де це може зіграти важливу роль. Якщо програма прийде до логічного кінця швидше й буде затрачено менше пам’яті, то результат вважатиметься позитивним.

2.2 Розробка методу визначення ефективності патернів

До програми по черзі будуть завантажуватися два файли: код з використанням патернів прив’язки даних, що вирішує конкретну задачу, а також код

без патернів, який вирішує ту саму задачу. Розроблювальний метод показаний у виді блок-схеми (рис. 2.1).

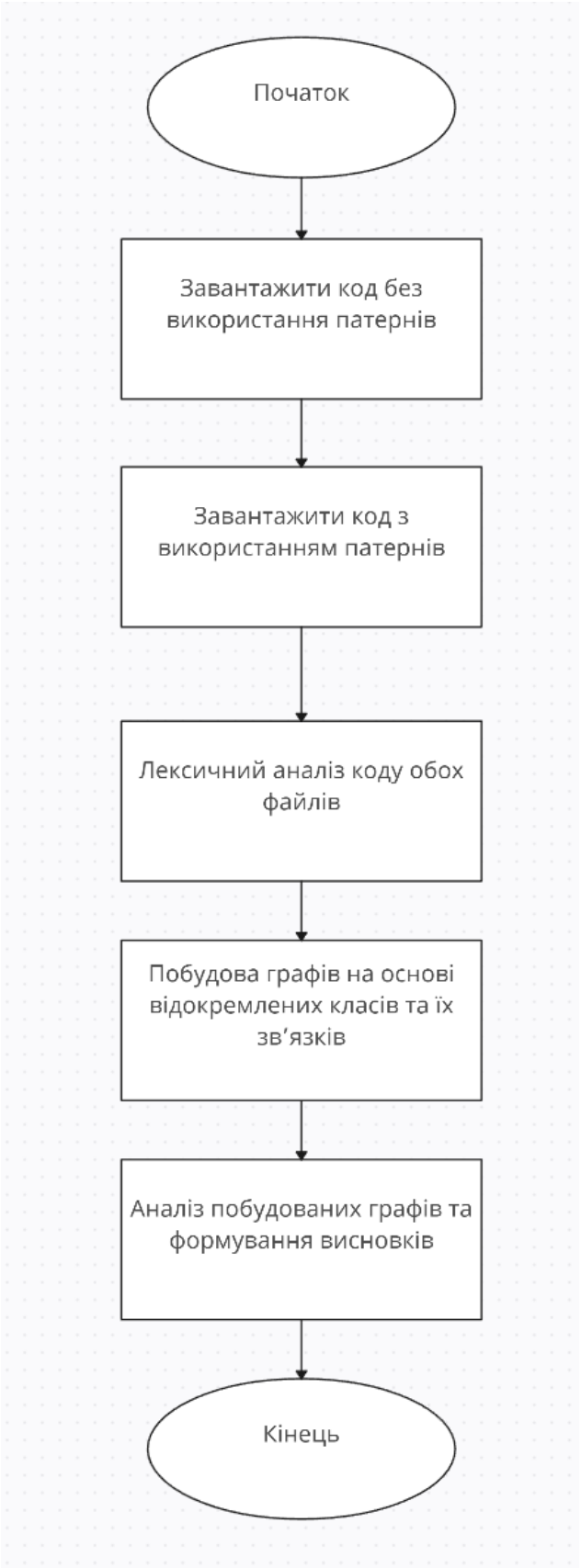

Рисунок 2.1 – Блок-схема роботи методу

Аналіз побудованих графів та формування висновків передбачає:

- порівняння кількості зв’язків у обох графах. Чим більше зв’язків, тим гірше. Очікуваний результат графу, побудованого за текстом програми з патернами прив’язки даних, має містити у собі менше зв’язків, ніж у першому графі;
- чим більше вага, тим гірше. Очікуваний результат графу з патернами – зменшення ваги кожного ребра, якщо це можливо;
- висновки формуються на основі визначених критеріїв (кількість зв'язків між класами в обох досліджуваних програмах, кількість рядків програмного коду, швидкодія цих програм, що вимірюється часом, за який програма дійде до логічного кінця, а також затрата апаратних ресурсів, що вимірюється кількістю затраченої пам’яті на роботу програми).

Детально приклад очікуваних змін розписано у таблиці (табл. 2.1).

Таблиця 2.1 – Таблиця очікуваних результатів перевірки

Критерій оцінювання	Очікуваний результат
Кількість зв’язків між класами у графі	Зменшилась у всіх, або декількох класів
Кількість рядків програмного коду	Зменшились, зменшились незначно (2-3 рядки), або залишились без змін
Швидкодія програми	Зменшилась незначно, або залишилась без змін
Затрата апаратних ресурсів	Зменшилась незначно, або залишилась без змін
Вага кожного ребра	Зменшилась між всіх класів, зменшилась між декількох класів, залишилась без змін між всіма класами (або зменшилась у одному)

Також окремо розроблено блок-схему роботи лексичного аналізатора (рис. 2.2).

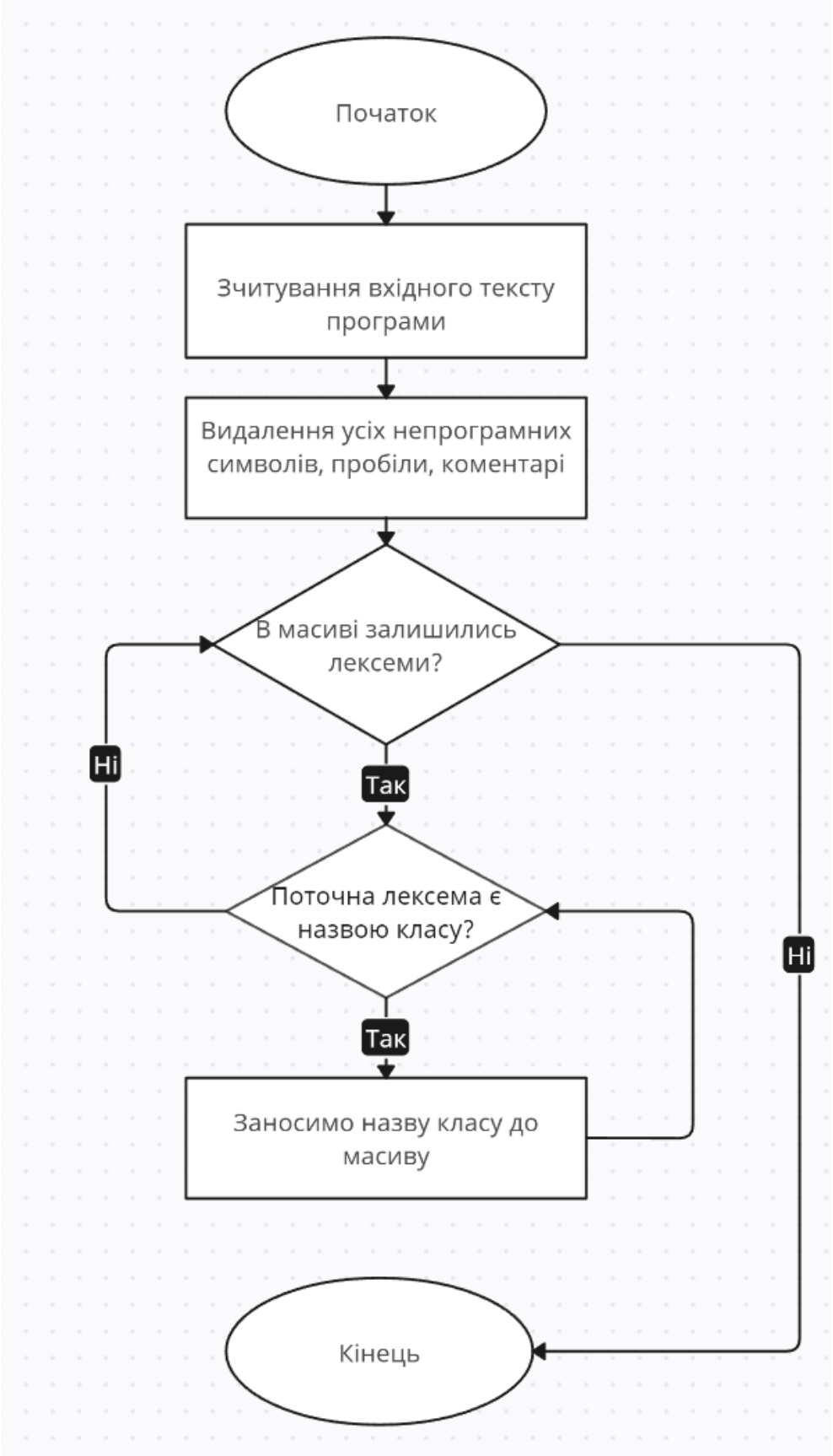

Рисунок 2.2 – Блок-схема лексичного аналізу коду

Видалення символів під час лексичного аналізу тексту програм передбачає такі дії:

- видалення усіх коментарів (одно та багато строкових);
- видалення усіх строкових літералів;
- представлення назв класів тексту програм, що перевіряються у вигляді масиву лексем.

Коли обидва файли пройдуть лексичний аналіз, будуть відокремлені програмні класи, а також зв’язки між ними, що додатково будуть зберігатися після лексичного аналізу тексту програм.

На основі цих даних будуть побудовані графи, які покажуть користувачу наглядно архітектуру програм. Вершинами графу виступатимуть класи, а ребра – зв’язки між цими класами. Вага ребра – кількість звертань одного класу до іншого. Направленість ребра показуватиме який клас до якого конкретно звертається.

Приклад подібного графу приведено у теоретичному прикладі (табл. 2.2) та візуалізації (рис. 2.3).

Таблиця 2.2 – Таблиця зв’язків класів

Назва класу	Кількість унікальних зв’язків	Взаємодія
A	3	C → A, A → B, B → A
B	0	B → A, A → B
C	1	C → D, C → A
D	0	C → D

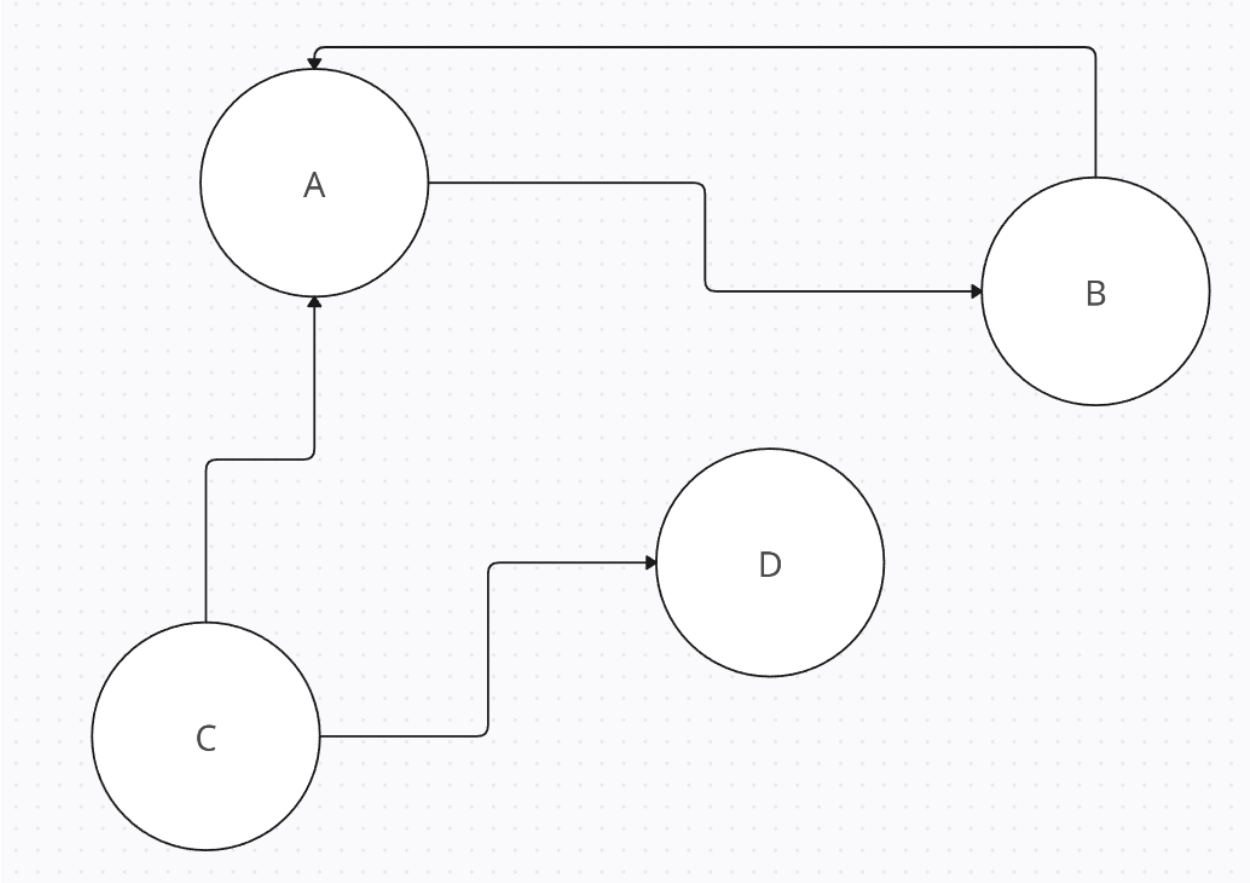

Рисунок 2.3 – Приклад графу

Порівняння двох графів:

- отримати кількість ребер у кожному графі;
- порівняти отримані значення.

Якщо кількість ребер в обох графах збігається, то графи вважаються еквівалентними у цьому аспекті. Інакше, можна сказати, що графи різняться за рівнем зв'язності.

Порівняння за вагою ребер:

- якщо графи містять зважені ребра, то отримати сумарну вагу всіх ребер у кожному графі;
- порівняти отримані значення;
- якщо сумарна вага ребер в обох графах збігається, графи вважаються еквівалентними в цьому аспекті. Інакше, можна сказати, що графи різняться за вагою своїх ребер.

Загальний принцип:

- якщо обидва аспекти (кількість зв'язків і вага ребер) збігаються, то графи вважатимуться еквівалентними з погляду обраного порівняння;
- якщо хоча б один із аспектів відрізняється, то графи різні за критерієм, що розглядається.

Приклад подібного порівняння приведено в окремих прикладах (рис. 2.4-2.5) та розібрано аналітично (табл. 2.3-2.5).

Таблиця 2.3 – Таблиця зв’язків для першого графу

Назва класу	Кількість унікальних зв’язків	Взаємодія та вага ребер
A	2 + 2 + 3 = 7	C → A (2), A → B (2), B → A (3)
B	0	B → A (3), A → B (2)
C	1 + 0 = 1	C → D (1), C → A (2)
D	0	C → D (1)

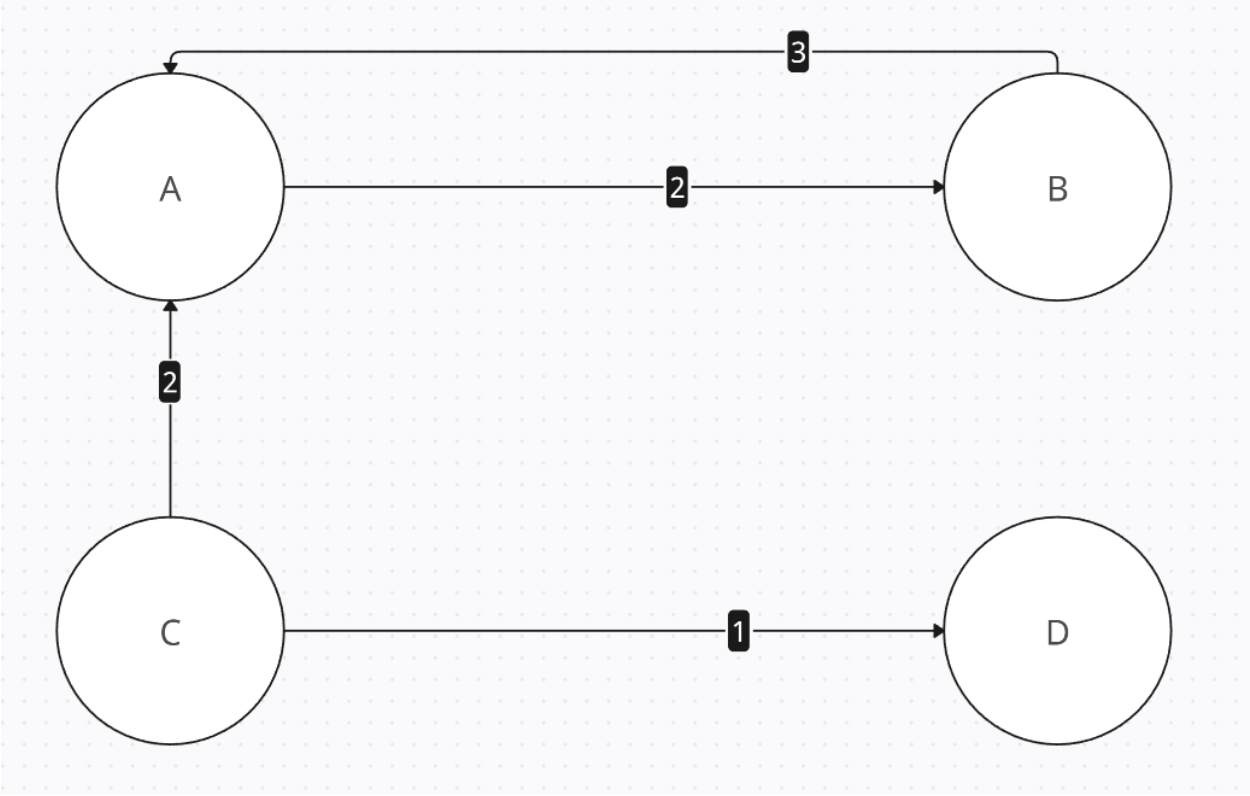

Рисунок 2.4 – Граф без використання патернів

Таблиця 2.4 – Таблиця зв’язків для другого графу

Назва класу	Кількість унікальних зв’язків	Взаємодія та вага ребер
A	1 + 1 +1 = 3	C → A (1), A → B (1), B → A (1)
B	0	B → A (1), A → B (1)
C	0	C →A (1)
D	0	Відсутні

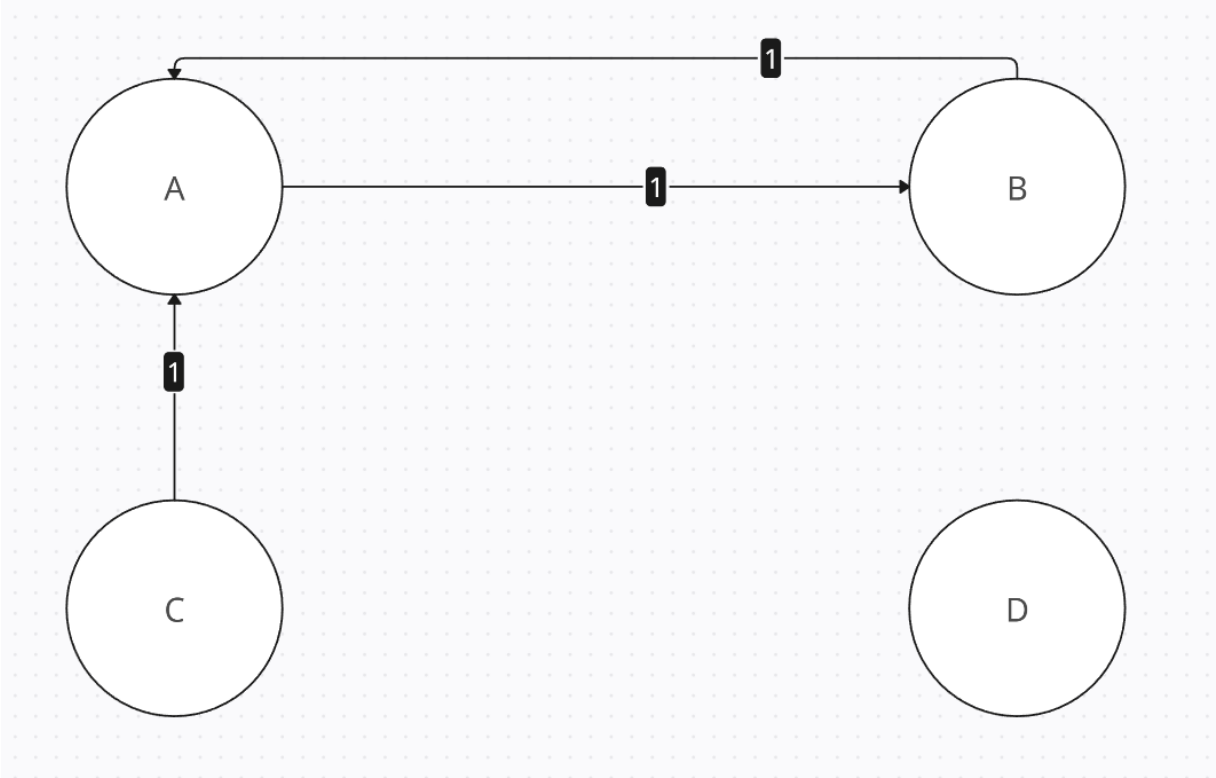

Рисунок 2.5 – Граф з використанням патернів

Таблиця 2.5 – Порівняння двох графів

Критерій	Перший граф	Другий граф
Кількість ребер	4	3
Кількість зв’язків класу A	7	3
Кількість зв’язків класу B	5	2

Продовження таблиці 2.5

Критерій	Перший граф	Другий граф
Кількість зв’язків класу C	3	1
Кількість зв’язків класу D	1	0
Загальна кількість зв’язків (сума ваги ребер)	3 + 2 + 2 + 1 = 8	1 + 1 + 1 = 3

За даною таблицею порівняння можна зробити такі висновки:

- у кожному з класів зменшилась кількість зв’язків. Зв’язки представлені не лише ребрами, а й їх вагою. Якщо вага ребра становить 2, то це означає, що один клас звертається до іншого два рази. Тому, якщо кількість таких звертань зменшилась, результат стає позитивним;
- загальна кількість зв’язків зменшилась, це позитивно впливає на архітектуру, тому в такому випадку можна зробити висновок, що патерни прив’язки даних позитивно вплинули на архітектуру, що свідчить про ефективність їхнього використання;
- кількість зв’язків також впливає на швидкодію програми. Якщо кількість зв’язків між класами зменшується, то зменшується й час, за який програма обробить усі запити й дійде до свого логічного завершення. Проте, також можливі ситуації, коли без патерну можна звертатися від класу до класу напряму, а тепер можливий транзит, що погіршить показник швидкодії;
- кількість рядків програмного коду теж може змінюватися, оскільки при зменшенні кількості зв’язків, також зменшується кількість рядків коду зі зверненнями. Проте, це більш ймовірно у великих програмах. У програмах більш меншого розміру ймовірне зменшення коду може сягати приблизно на 20% (зменшення кількості класів тощо).

Висновки до розділу 2

У даному розділі було розроблено метод, що дозволяє на основі використання елементів теорії графів визначати ефективність використання патернів прив’язки даних у конкретних задачах. Цей метод за допомогою лексичного аналізу тексту програм, що завантажуються до додатку, перетворює класи у орієнтований граф, що показує структурні зв’язки між цими класами. За результатами опубліковано роботу [2].

РОЗДІЛ 3 ПРОЄКТУВАННЯ Й РОЗРОБКА ІНСТРУМЕНТАЛЬНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ВИЗНАЧЕННЯ ЕФЕКТИВНОСТІ ПАТЕРНІВ ПРИВ'ЯЗКИ ДАНИХ

3.1 Зовнішнє проєктування

Для реалізації запропонованого методу дослідження патернів прив'язки даних за зчепленням був розроблений програмний додаток. В даному розділі наведено основні етапи його проєктування.

3.1.1 Вхідні дані

Вхідними даними програми є:

– тексти програм на мові C#;

– назва та шлях до директорії з проєктом;

– файли з розширенням .CS.

3.1.2 Вихідні дані

Результатом роботи програми є наступні вихідні дані:

– матриці суміжності та граф, які відображають структурні зв'язки між класами, які визначені лексичним аналізом коду;

– текстове повідомлення з висновком щодо результатів оцінки використання патернів у даній програмі;

– текстові повідомлення, щодо успішності або неуспішності виконаних операцій роботи програми.

3.1.3 Функціональні вимоги

– завантаження з файлу тексту програми;

– завантаження до програми файл коду з директорії;

– побудова графу на основі матриці суміжності, що показує зв'язки між класами;

– надання користувачеві програмного продукту висновку щодо результатів дослідження ефективності використання патернів прив'язки даних в обраному коді програми;

– лексичний аналіз тексту програм, що полягає у виділенні назв класів програм, а також їх зв’язків.

3.2 Базова архітектура системи

3.2.1 Сценарій роботи програми

- після відкриття програми у користувача з’являється можливість прочитати можливості програми, або ж завантажити файл, з програмним кодом на мові C#, що містить класи;
- після цього він натискає кнопку «Побудувати граф по матриці».
- програма робить лексичний аналіз коду і виділяє класи як програмні сутності, заповнює матрицю, де 1 – це присутність класу, а по стовпчиках 1 – є зв’язок, 0 – зв’язка між класами немає;
- по цій матриці будується граф, що візуально показує архітектуру коду. Вершина графу – це клас, ребра – це зв’язки між ними. Вага ребра – це кількість звертань класу до іншого класу;
- на основі отриманих результатів зв’язків, програма записує результати перевіреного коду за зчепленням у файл;
- далі користувачу пропонується завантажити додатковий файл з кодом, але з використанням патернів;
- програма повторює цикл, як з першим варіантом;
- далі користувач натискає кнопку «Порівняти результати», йому надається висновок, що містить результати зчеплення в архітектурі в обох випадках, а також надаються числові значення: кількість зв’язків, кількість звертань між класами.
- на основі цього можна буде оцінити ефективність роботи патерну у даній задачі.

3.2.2 Етапи методу

- завантажуються файли з кодом на мові C#: один буде з використанням патернів, інший – без. Програми вирішують одну й ту саму задачу;
- робиться лексичний аналіз коду обох файлів, який розбере текст програми на класи і відслідкує зв’язки між ними;

- програма створює текстовий файл, в який записується кількість зв’язків та звертань між класами у першому й другому файлах з кодом;
- йде лексичний аналіз коду, на основі чого будується граф. Де вершини – це класи, а ребра – зв’язки;
- визначається ступінь зв’язності графу;
- генеруються висновки на основі перевірки двох файлів, а також порівняння двох програм за отриманими графами.

Для більш детального розуміння функціональних можливостей програми також було розроблено окреме моделювання кожного процесу (рис. 3.1 – 3.4).

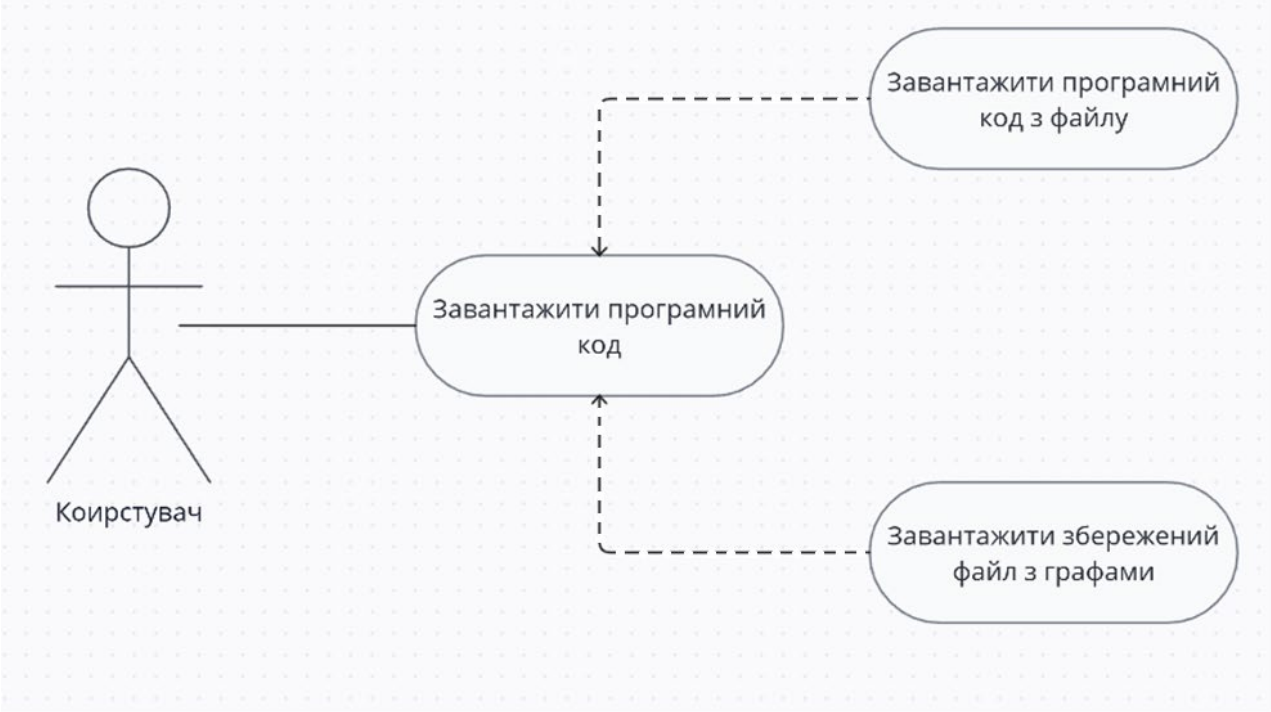

Рисунок 3.1 – Діаграма прецедентів введення вхідних даних

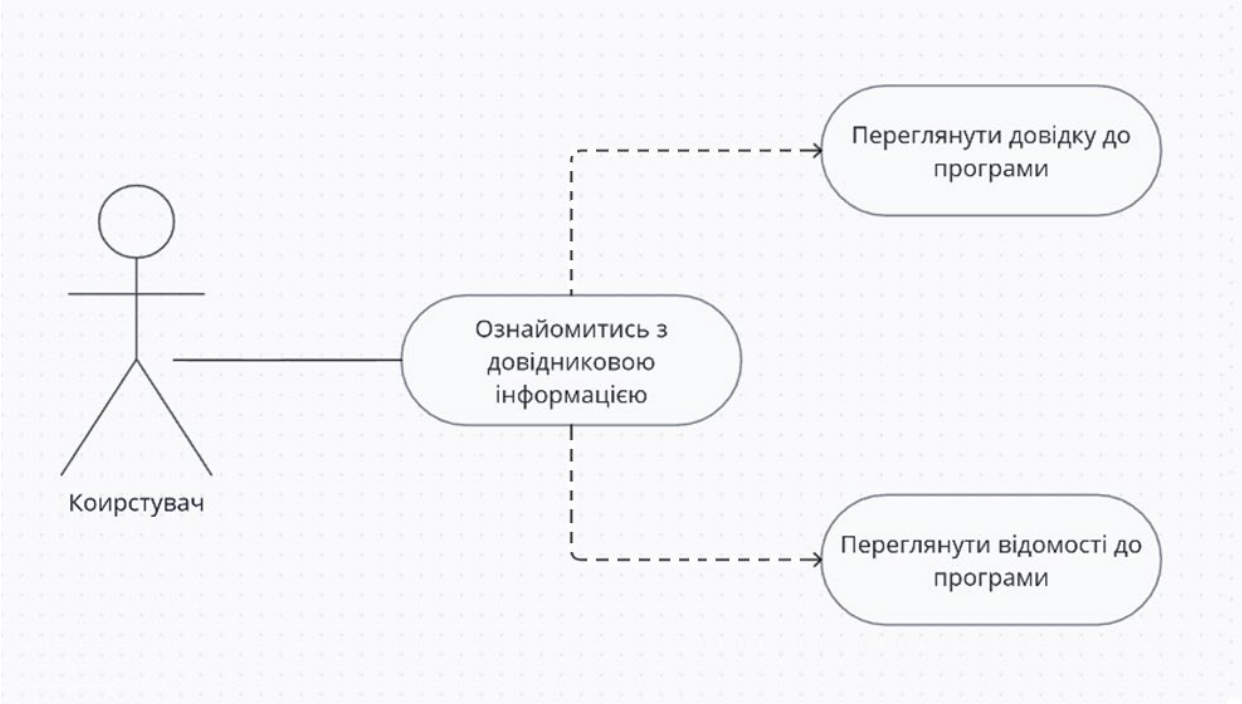

Рисунок 3.2 – Діаграма прецедентів ознайомлення з довідками

Рисунок 3.3 – Діаграма прецедентів для визначення ефективності

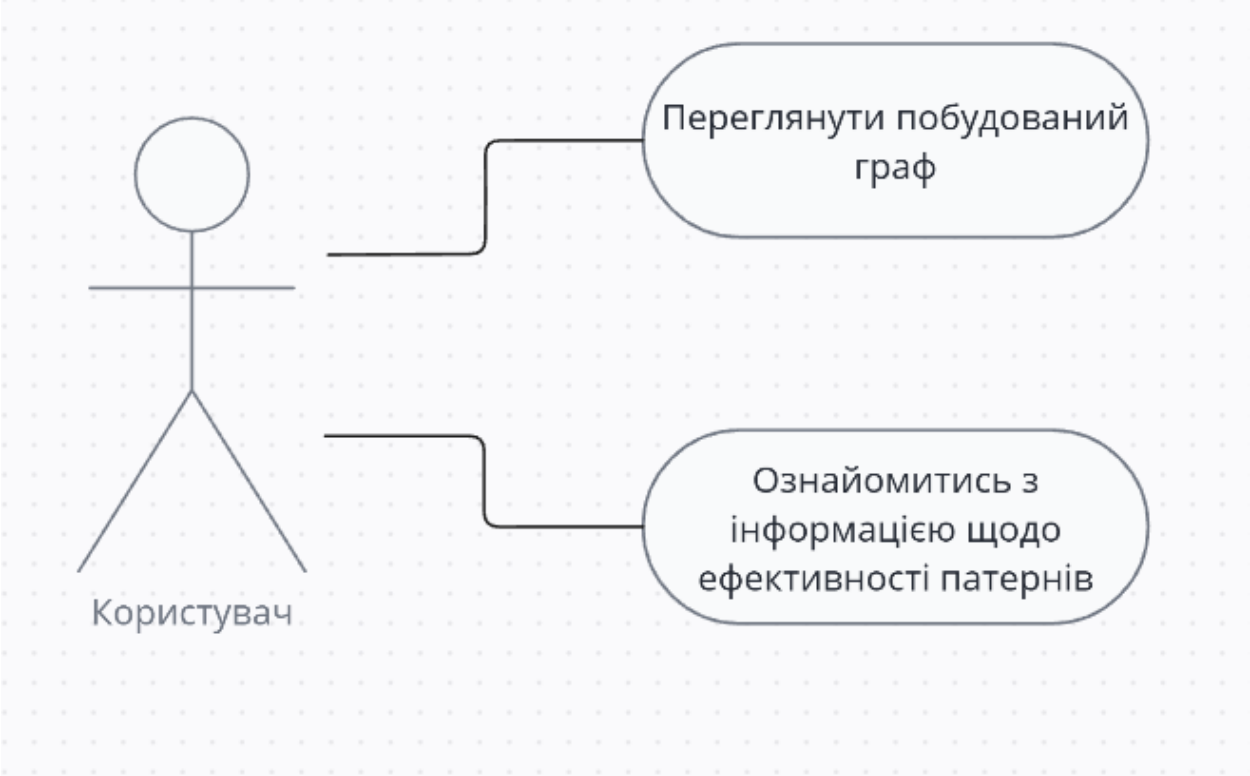

Рисунок 3.4 – Діаграма прецедентів для перегляду вихідних даних

3.3 Внутрішнє проєктування

У розроблювальній програмі класи було поділено на три рівні: логіки, даних та представлення. Рівень логіки класів охоплюють способи організації інформації та мислення в інформаційних системах. На рівні логіки розглядаються принципи логічного виведення та обробки даних, що забезпечують логічну відповідність та точність у вирішенні завдань. Рівень даних у даній роботі представляє собою клас, що містить методи для завантаження вхідних даних з форми [4]. Рівень представлення включає способи візуалізації та структурування даних для ефективного сприйняття та взаємодії, що сприяє кращому розумінню та прийняттю рішень.

Рівень логіки даної програми включає в себе такі класи:

- Graph – відповідає за побудову графа;
- LexicalAnalysis – відповідає за лексичний аналіз коду;
- ClassConnections – відповідає за зв’язки між класами у коді;
- Preliminary – відповідає за попередню обробку коду;
- Efficiency – має методи, що визначають ефективність патернів;

Рівень представлення включає в себе наступні класи:

- MainForm – відповідає за головну форму програми;
- ResultForm – відповідає за додаткову форму з результатами визначення ефективності використання патернів;

Рівень даних включає в себе один клас InputData, який відповідає за методи, що завантажують вхідні дані.

Специфікація визначених класів у CRC-картках (табл. 3.1 – 3.8):

Таблиця 3.1 – CRC-картка класу Graph

Базовий клас	Похідні класи (нащадки)
–	–
Обов’язки	Зв’язки
Створення графу на основі отриманих результатів після лексичного аналізу	ClassConnections, MainForm, Efficiency

Таблиця 3.2 – CRC-картка класу InputData

Базовий клас	Похідні класи (нащадки)
–	–
Обов’язки	Зв’язки
Завантаження у додатку вхідних даних	MainForm

Таблиця 3.3 – CRC-картка класу LexicalAnalysis

Базовий клас	Похідні класи (нащадки)
–	–
Обов’язки	Зв’язки
Відповідає за лексичний аналіз завантаженого коду	MainForm, Preliminary, ClassConnections

Таблиця 3.4 – CRC-картка класу ClassConnections

Базовий клас	Похідні класи (нащадки)
–	–
Обов’язки	Зв’язки
Відповідає за створення та збереження списку зв’язків між класами в розібраному тексті програми	Graph, LexicalAnalysis, MainForm

Таблиця 3.5 – CRC-картка класу Preliminary

Базовий клас	Похідні класи (нащадки)
–	–
Обов’язки	Зв’язки
Відповідає за попередню обробку коду	LexicalAnalysis

Таблиця 3.6 – CRC-картка класу Efficiency

Базовий клас	Похідні класи (нащадки)
–	–
Обов’язки	Зв’язки
Визначення ефективності використання патернів	Graph, MainForm

Таблиця 3.7 – CRC-картка класу MainForm

Базовий клас	Похідні класи (нащадки)
–	–
Обов’язки	Зв’язки
Відповідає за роботу головної форми програми	Graph, InputData, LexicalAnalysis, ClassConnections, Preliminary, Efficiency, ResultForm

Таблиця 3.8 – CRC-картка класу ResultForm

Базовий клас	Похідні класи (нащадки)
–	–
Обов’язки	Зв’язки
Робота з формою результатів визначення ефективності використання патернів	MainForm

На основі визначених класів та опису їх обов’язків та зв’язків було побудовано діаграму класів (рис. 3.5).

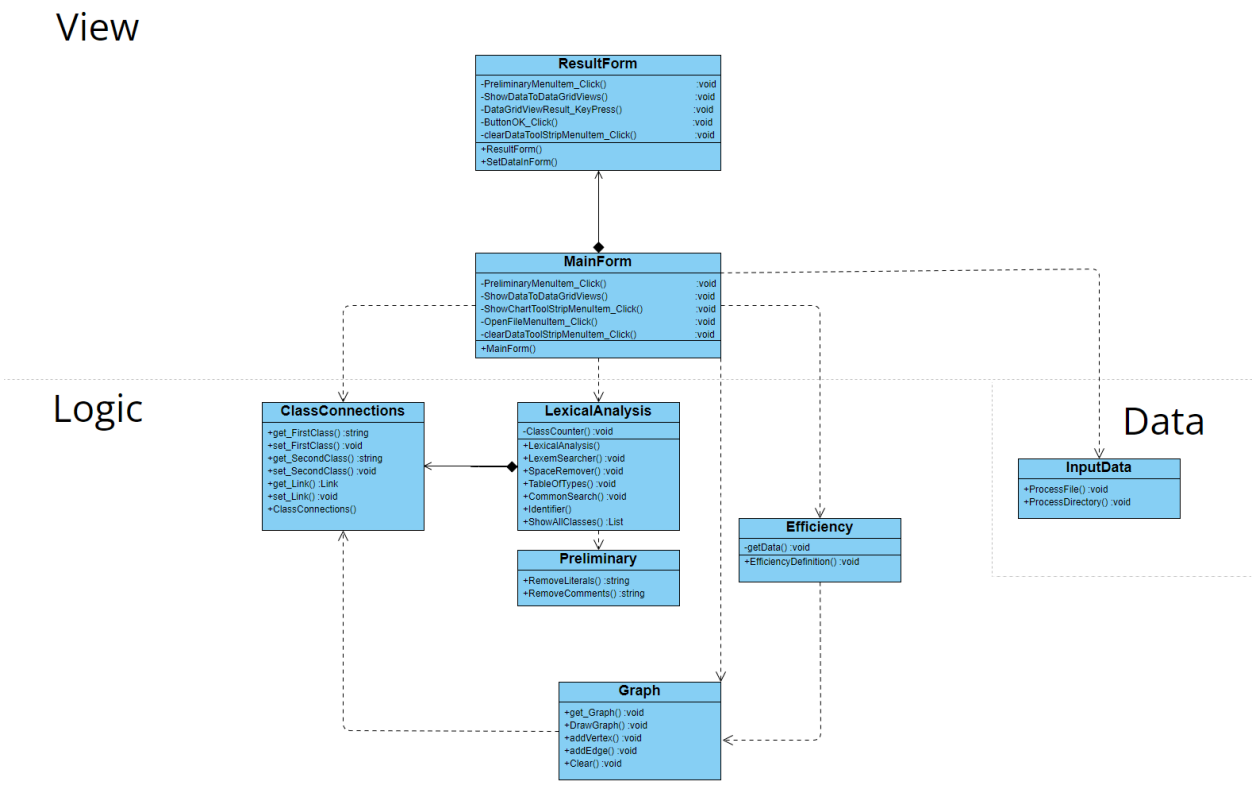

Рисунок 3.5 – Діаграма класів

3.4 Розробка інтерфейсу користувача

Після визначення класів та вимог до програми, було розроблено інтерфейс додатку. Він буде розроблений за допомогою засобів Windows Forms, що дозволить легко зробити інтерфейс з чотирма вікнами [5]. В інтерфейс користувача входять наступні форми:

1) Головна форма програми (рис. 3.6). Вона містить у собі такі компоненти:

- секції з матрицями побудованих графів;
- секції обох графів;
- головне меню;
- допоміжне меню «Файл», «Довідка», «Інструменти».

2) Форма довідкової інформації (рис. 3.7). Вона буде містити у собі теоретичну інформацію, що допоможе користувачу більш детально розібратися з суттю додатку;

3) Форма інформації про програму (рис. 3.8). Вона буде містити у собі інформацію про розробника додатку, актуальну версію, а також про основне призначення програми;

4) Форма з результатами дослідження (рис. 3.9).

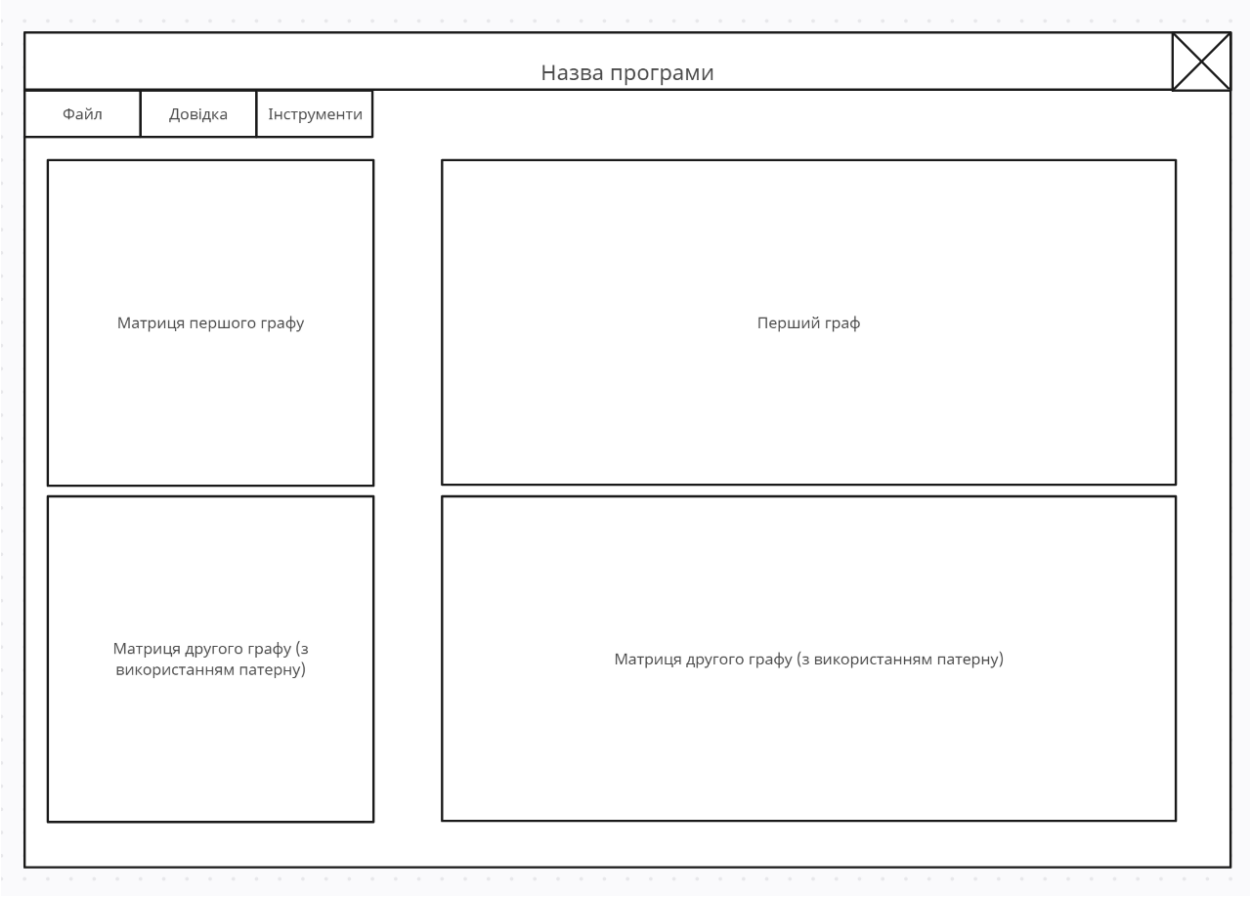

Рисунок 3.6 – Головна форма програми

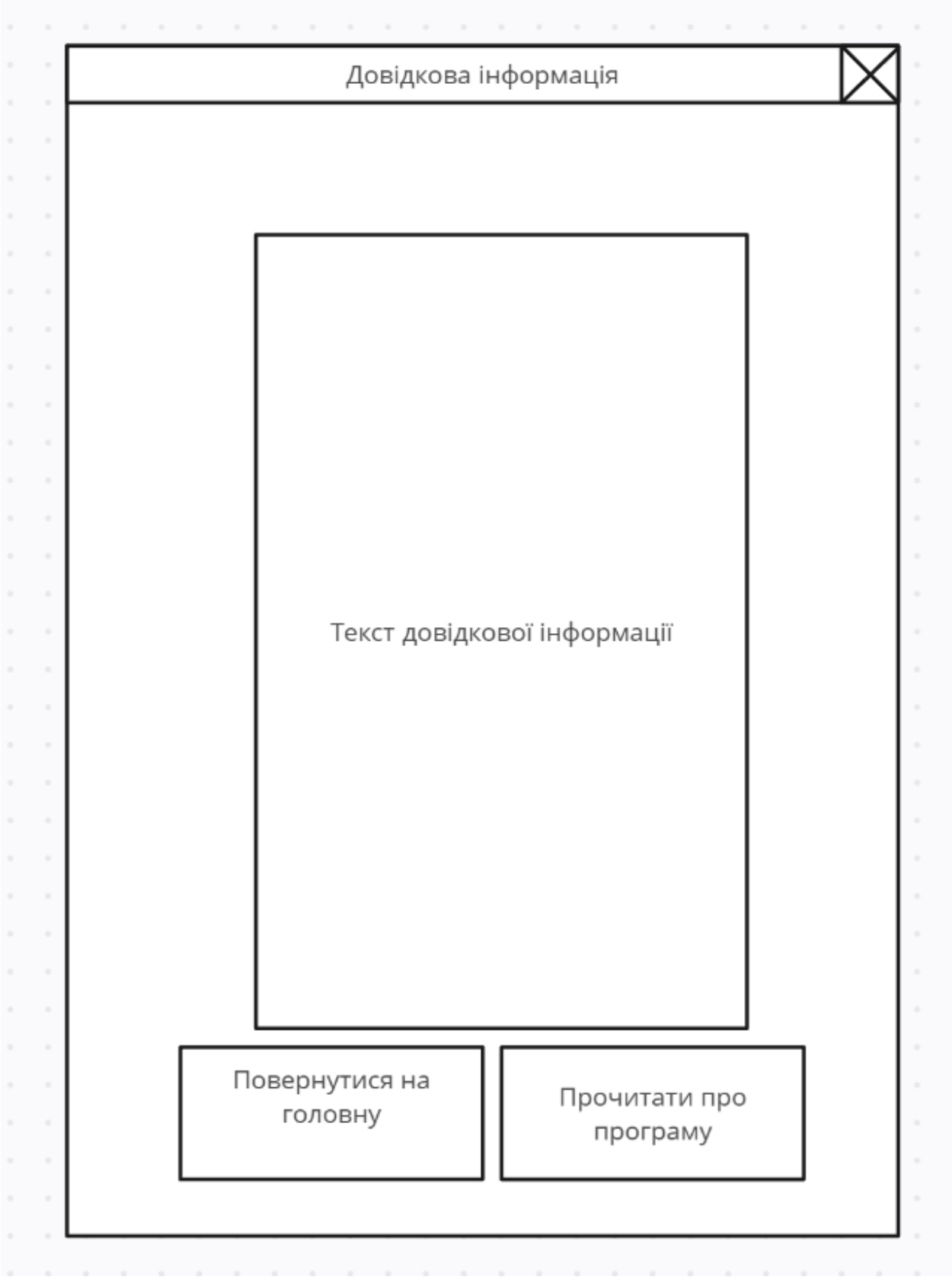

Рисунок 3.7 – Форма довідкової інформації

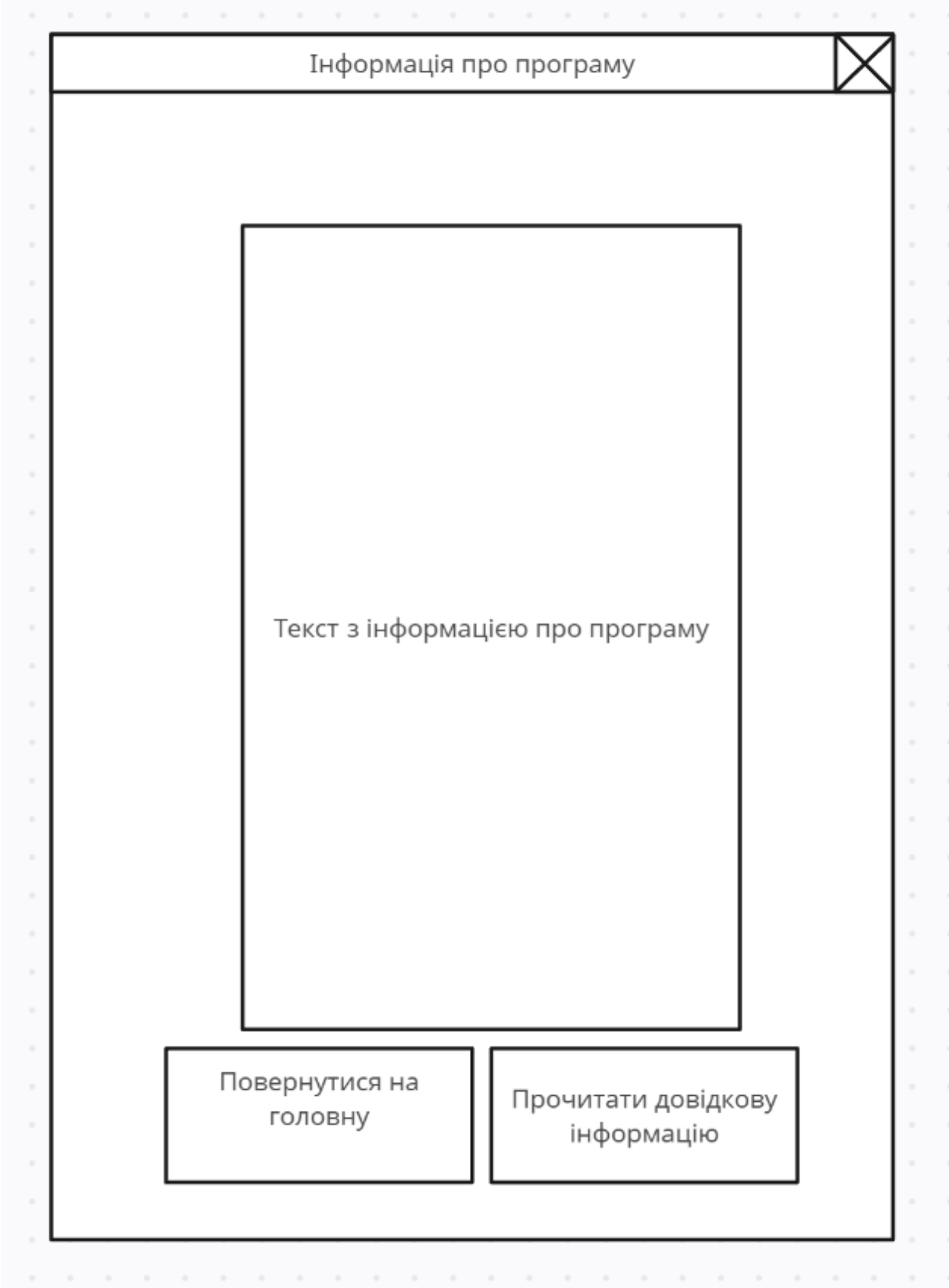

Рисунок 3.8 – Форма інформації про програму

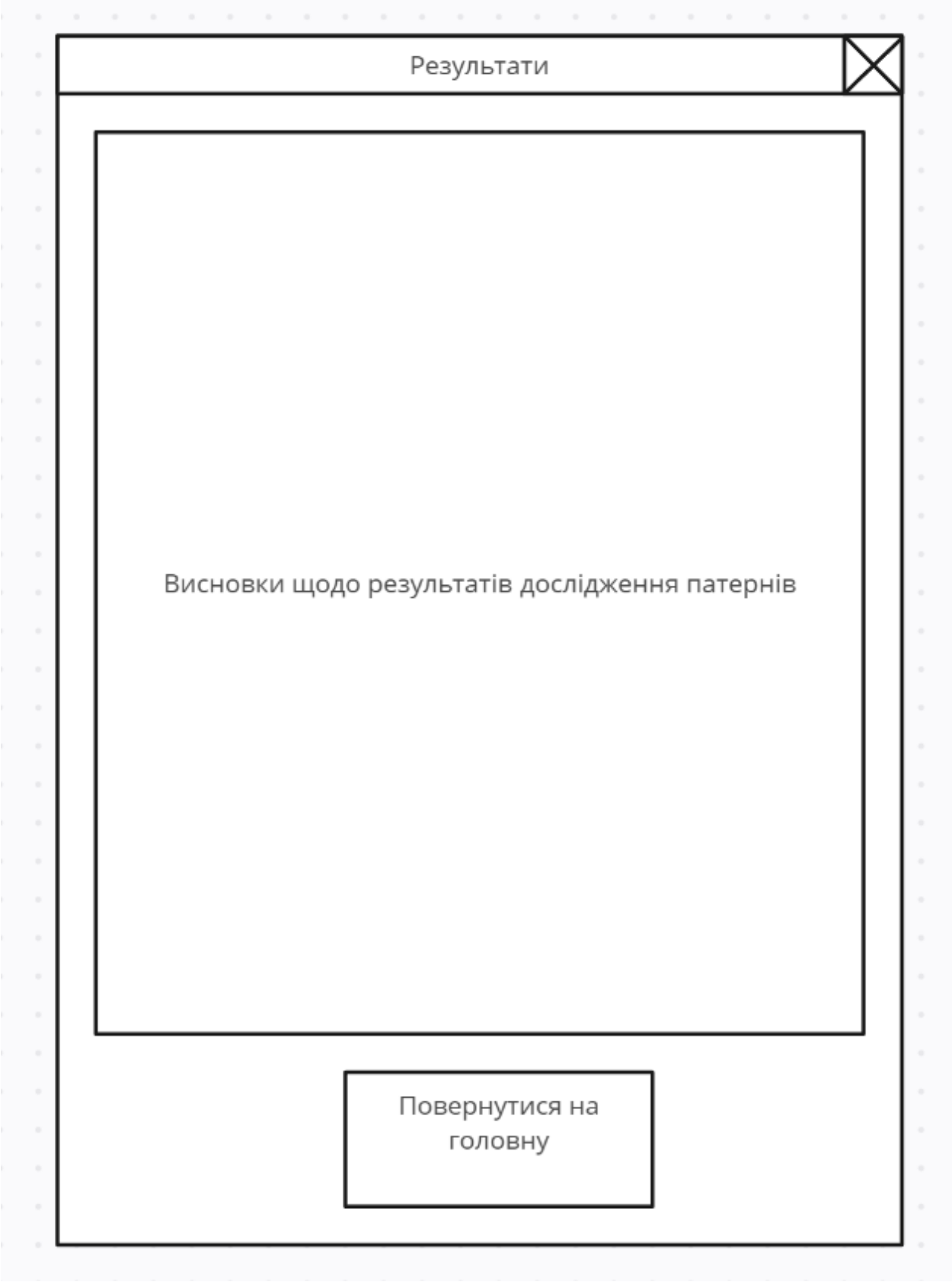

Рисунок 3.9 – Форма з результатами дослідження

Висновки до розділу 3

У даному розділі було розібрано ключові моменти проєктування програмного додатку для визначення ефективності використання патернів прив’язки даних за зчепленням у окремих задачах. Було визначено класи й розглянута їх структура та взаємодія один з одним за допомогою CRC-карток, діаграми класів та діаграми послідовностей. Спроєктовано інтерфейс користувача, розроблено ескізи усіх форм, а також зроблено їх опис.

РОЗДІЛ 4 ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ПАТЕРНІВ ПРИВ'ЯЗКИ ДАНИХ

4.1 Підготовка до експерименту

Для проведення дослідження було визначено основні критерії визначення ефективності патернів прив'язки даних:

- кількість зв'язків між класами у програмі з використанням патернів та без них у одній і тій самій задачі;
- кількість самих класів у обох випадках;
- швидкодія програм у обох випадках. Вона вимірюється тим, за який час кожна програма дійде до свого логічного кінця за один програмний цикл;
- затрата апаратних ресурсів, що вимірюється кількістю затраченої пам'яті на роботу обох програм;
- кількість рядків програмного коду у обох програмах.

Етапи проведення експерименту:

- побудова діаграми класів та графу за допомогою програм для першого та другого коду;
- табличне представлення отриманих даних й їх огляд;
- створення висновків за допомогою розробленого додатку;
- формування результатів експерименту в окремій таблиці.

Вхідними даними будуть файли з кодом програм. Для підготовки до експерименту було обрано одну задачу, що моделює систему «Бібліотека». У першому випадку вона реалізована кодом на мові C# (програмний код 1). У другому випадку вона реалізована з використанням патернів прив'язки даних (програмний код 2). До обох кодів також представлено їх діаграми класів (рис. 4.1-4.2).

Програмний код 1 (без патернів прив'язки даних):

```
using System;
using System.Collections.Generic;

// Клас, представлюючий книгу
```

```
public class Book
{
    public string Title { get; set; }
    public string Author { get; set; }
    public int Year { get; set; }
    public string BookGenre { get; set; }
}

// Клас, представлюючий бібліотеку
public class Library
{
    private List<Book> books;

    public Library()
    {
        books = new List<Book>();
    }

    public void AddBook(Book book)
    {
        books.Add(book);
    }

    public List<Book> GetBooks()
    {
        return books;
    }
}

// Клас, представлюючий читача
public class Reader
{
    public string Name { get; set; }

    public void BorrowBook(Book book)
    {
        Console.WriteLine($"{Name} borrowed the book '{book.Title}' by {book.Author}");
    }
```

```
    public void ReturnBook(Book book)
    {
        Console.WriteLine($"{Name} returned the book '{book.Title}' by {book.Author}");
    }
}

// Клас, представлюючий бібліотекаря
public class Librarian
{
    public void CheckIn(Book book)
    {
        Console.WriteLine($"Checked in the book '{book.Title}' by {book.Author}");
    }

    public void CheckOut(Book book)
    {
        Console.WriteLine($"Checked out the book '{book.Title}' by {book.Author}");
    }
}

// Клас, представлюючий жанр книги
public class Genre
{
    public string Name { get; set; }
}

class Program
{
    static void Main()
    {
        // Використання класів
        Book book1 = new Book { Title = "Земля", Author = "Ольга Кобилянська", Year = 1902 };
        Book book2 = new Book { Title = "Чорна рада", Author = "Пантелеймон Куліш", Year = 1857 };

        Library library = new Library();
        library.AddBook(book1);
        library.AddBook(book2);
```

```
        Reader reader = new Reader { Name = "Василій Петров" };
        reader.BorrowBook(book1);

        Librarian librarian = new Librarian();
        librarian.CheckOut(book2);
    }
}
```

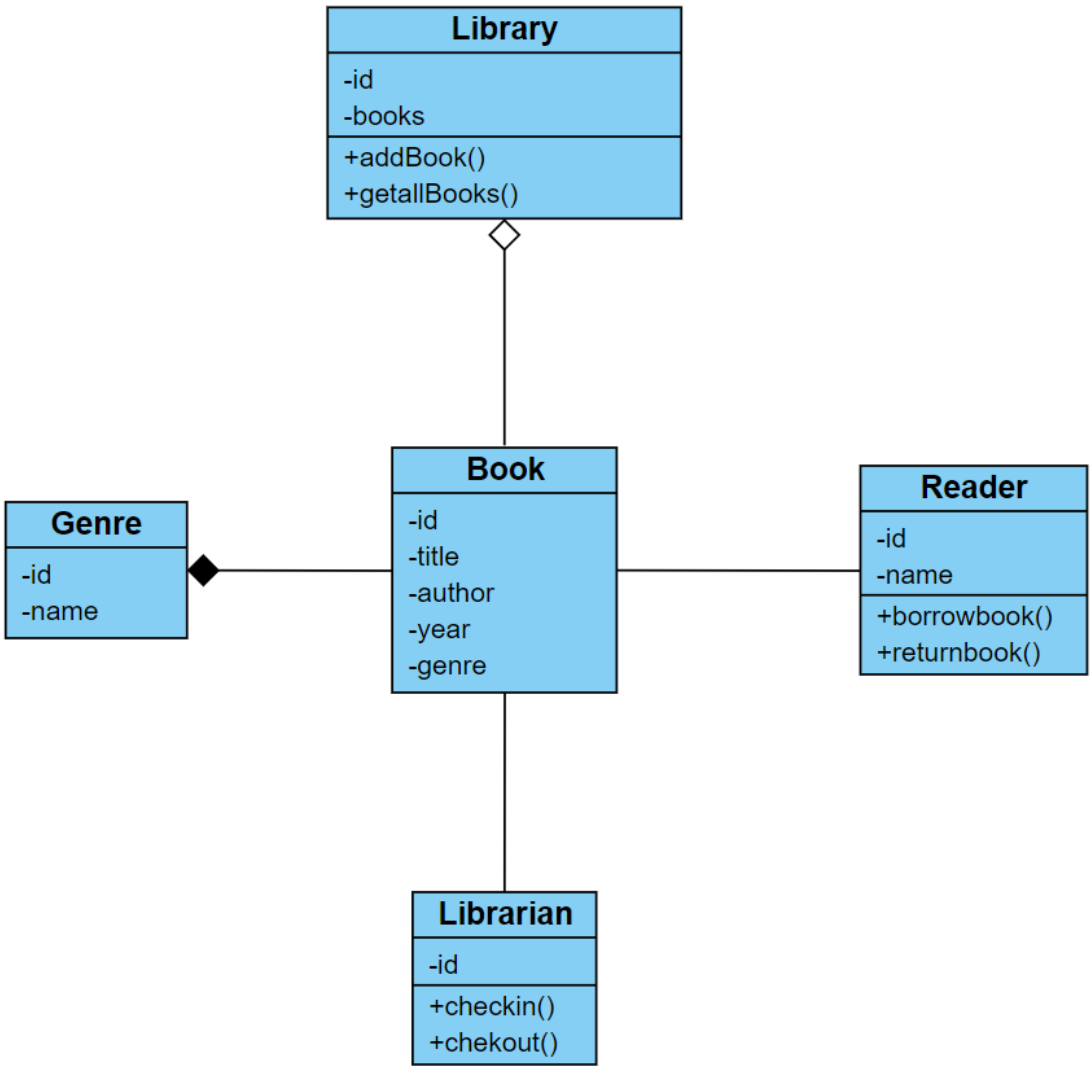

Рисунок 4.1 – Діаграма класів першого коду

Програмний код 2 (з використанням патернів прив’язки даних):

```
using System;
using System.Collections.ObjectModel;
```

```
public class Book
{
    public string Title { get; set; }
    public string Author { get; set; }
    public int Year { get; set; }
    public string BookGenre { get; set; }
}

public class LibraryViewModel
{
    public ObservableCollection<Book> Books { get; set; } = new ObservableCollection<Book>();
    public string UserName { get; set; }

    public void BorrowBook(Book book)
    {
        Console.WriteLine($"{UserName} borrowed the book '{book.Title}' by {book.Author}");
    }

    public void ReturnBook(Book book)
    {
        Console.WriteLine($"{UserName} returned the book '{book.Title}' by {book.Author}");
    }

    public void CheckIn(Book book)
    {
        Console.WriteLine($"Checked in the book '{book.Title}' by {book.Author}");
    }

    public void CheckOut(Book book)
    {
        Console.WriteLine($"Checked out the book '{book.Title}' by {book.Author}");
    }
}

class Program
{
    static void Main()
    {
```

```
        LibraryViewModel libraryViewModel = new LibraryViewModel();

        Book book1 = new Book { Title = "Земля", Author = "Ольга Кобилянська", Year = 1902 };
        Book book2 = new Book { Title = "Чорна рада", Author = "Пантелеймон Куліш", Year = 1857 };

        libraryViewModel.Books.Add(book1);
        libraryViewModel.Books.Add(book2);

        libraryViewModel.UserName = "Василій Петров";
        libraryViewModel.BorrowBook(book1);

        libraryViewModel.CheckOut(book2);
    }
}
```

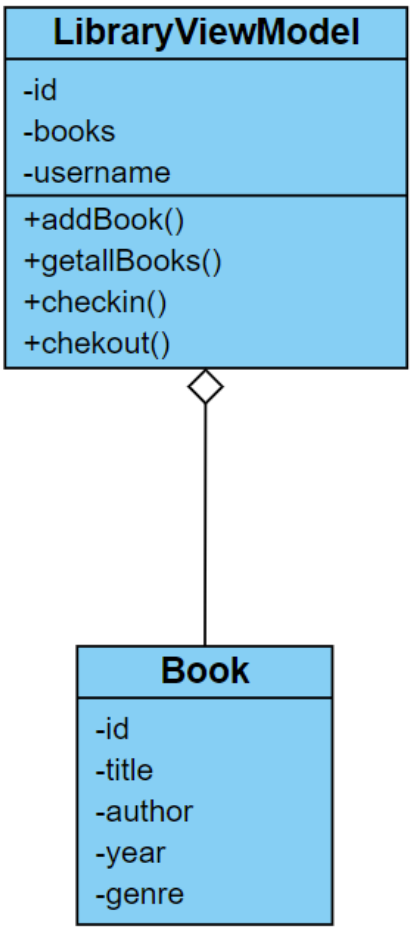

Рисунок 4.2 – Діаграма класів другого коду

4.2 Проведення експерименту

На основі цих кодів було побудовано два графи за допомогою розробленого додатку. Вершини – це програмні класи, а ребра – зв’язки між ними.

Перший граф (рис. 4.3):

Вершина 1: бібліотека містить книги, тому стрілка спрямована від бібліотеки до книги, вказуючи, що кожна книга належить певній бібліотеці.

Вершина 2: бібліотекар керує бібліотекою та відповідає за видачу книг, тому стрілка спрямована від бібліотекара до книги, відображаючи, що бібліотекар має вплив на статус та доступність кожної книги.

Вершина 3: читач бере книгу з бібліотеки, тому стрілка спрямована від читача до книги, показуючи, що читач використовує певну книгу для читання.

Вершина 4: книга, яка пов’язана з усіма іншими вершинами;

Вершина 5: кожна книга належить певному жанру, тому стрілка спрямована від жанру до книги, вказуючи, що кожна книга пов'язана з конкретним жанром.

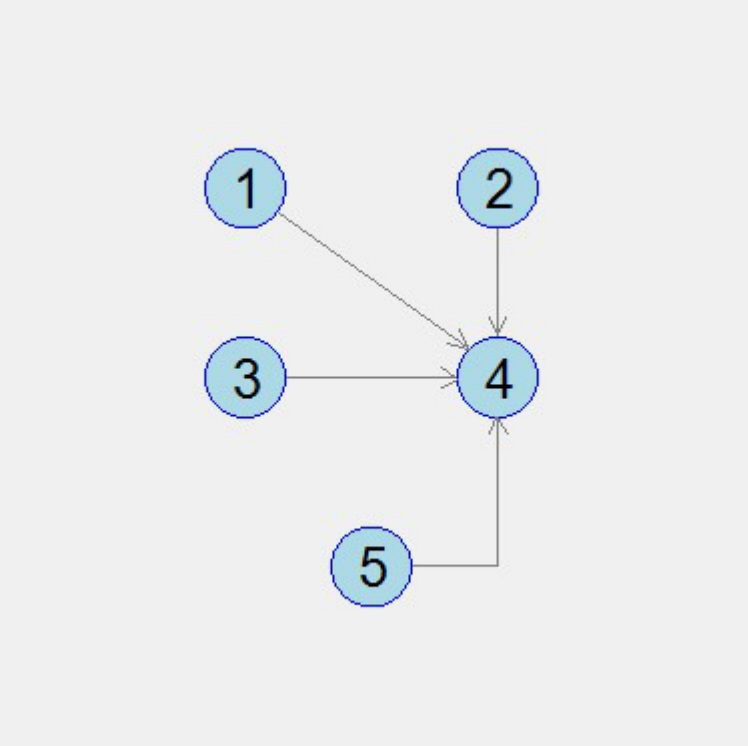

Рисунок 4.3 – Граф для першого коду

Другий граф (рис. 4.4):

Вершина 1: книга, яка пов’язана з основним класом;

Вершина 2: клас, що реалізований за допомогою патерну прив’язки даних, який містить у собі інформацію про усі елементи, що описані у вершинах 1-3, 5 першого графу.

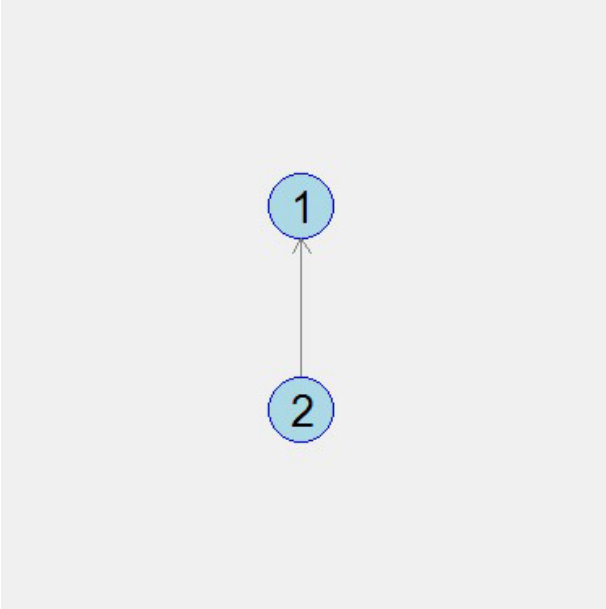

Рисунок 4.4 – Граф для другого коду

За даними цих графів побудуємо таблиці з характеристиками цих графів на основі програмного коду (табл. 4.1-4.2):

Таблиця 4.1 – Характеристики першого графу

Назва класу	Кількість унікальних зв’язків	Взаємодія
Library	1	Library → Book
Librarian	1	Librarian → Book
Genre	1	Genre → Book
Reader	1	Reader → Book
Book	0	Library → Book; Librarian → Book; Genre → Book; Reader → Book

Таблиця 4.2 – Характеристики другого графу

Назва класу	Кількість унікальних зв’язків	Взаємодія
LibraryViewModel	1	Book → → LibraryViewModel

Book	0	Book → → LibraryViewModel

За даними цих таблиць можна зробити такі висновки:

- кількість програмних класів зменшилась з п’яти до двох;
- кількість унікальних зв’язків між ними зменшилась з чотирьох до одного зв’язка;

Це позитивно вплинуло на архітектуру програми, тож патерни прив’язки даних в цьому випадку покращують роботу програми. Також, помітно зменшилась кількість рядків програмного коду в порівнянні з першою програмою, що свідчить про полегшення для розробника написання коду у подібних задачах.

4.3 Результати експерименту

Після розбору усіх вхідних даних, а також табличного представлення результатів порівняння графів, за допомогою розробленого додатку було сформовано висновки (рис. 4.5), в яких детально порівнюються обидва графи, а також програмний код програм.

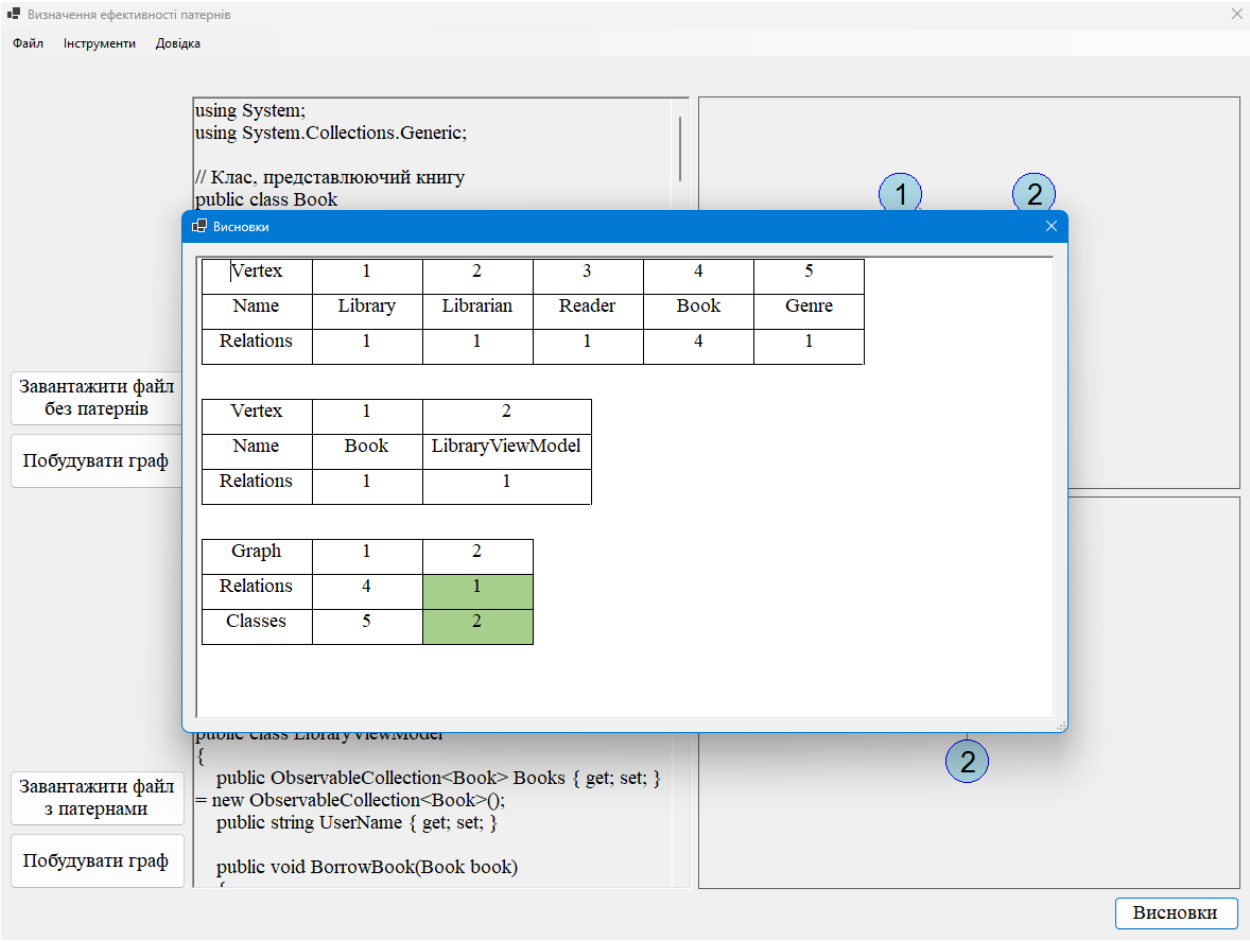

Рисунок 4.5 – Програмні висновки

Висновки до розділу 4

У четвертому розділі було описано етапи проведення експерименту з визначення ефективності патернів прив’язки даних. Спершу було сформульовано основні критерії визначення ефективності. Мова коду програмних файлів, що були взяті як вхідні дані, написані на мові програмування C#. Після цього було побудовано діаграми класів обох кодів, а також два графи, що були побудовані за допомогою програмного додатку. На основі цих даних було створено таблиці, в яких описуються унікальні зв’язки та взаємодії між програмними класами в обох випадках. Результати цих таблиць показали, що в одній і тій самій задачі, патерни допомогли зменшити кількість програмних класів, а також зменшити кількість унікальних зв’язків між ними, що позитивно вплинуло на архітектуру програми. За допомогою розробленого програмного додатку було створено висновки, що детально показують зміни у програмі. Результати експерименту окремо наведені в таблиці, що порівнює

отримані вихідні дані. Результати проведення експерименту можна вважати успішними, оскільки реальні порівняння співпадають з програмними Це говорить про те, що розроблений під час дослідження метод дозволяє визначати ефективність використання патернів прив’язки даних.

ВИСНОВКИ

В результаті роботи було розроблено метод визначення ефективності використання патернів прив’язки даних.

Було розглянуто існуючі підходи до визначення ефективності патернів, аналізу коду, а також методів створення графів. Повного програмного аналогу для визначення ефективності патернів немає, тому спочатку було сформовано критерії, за якими планується визначати ефективність, а також створено блок-схему роботи методу.

Під час розробки методу визначення ефективності використання патернів було визначено основні етапи: лексичний аналіз програмного коду обох файлів, що перевіряються, побудова графів на основі цих кодів, де класи – вершини графу, а ребра – зв’язки між ними, а також подальший аналіз цих графів, на основі яких окремо формуються висновки щодо впливу патернів на архітектуру.

Наступним етапом було формування вимог до розроблюваного програмного забезпечення, виконано зовнішнє на внутрішнє проєктування: створено основні діаграми, які визначили програмні сутності.

Заключним етапом дослідження стало проведення експерименту з визначення ефективності використання патернів, їх впливу на архітектуру. На основі цього було сформульовано висновки щодо працездатності розробленого методу. Результати дослідження показали, що патерни позитивно вплинули на архітектуру. Проте, для більш детальної перевірки методу слід визначати ефективність у більш великих програмах, з великою кількістю класів.

Розроблений програмний додаток може бути використаний викладачами кафедри КІТ, а також студентами кафедри під час написання коду, де можливий варіант використання патернів прив’язки даних.

СПИСОК ЛІТЕРАТУРИ

1. Тепляков С. Патерни проєктування на платформі .NET – СПб: Пітер, 2016 – 320 с.

2. Бажин В. А., «Дослідження патернів прив’язки даних за зчепленням» (кер. доц. Куроп’ятник О. С.) // Всеукраїнська науково-технічна конференція студентів і молодих учених “наука і сталий розвиток транспорту 2023” 27 жовтня 2023 року збірник тез том II / Міністерство освіти і науки України Український державний університет науки і технологій. – с. 27-28.

3. Фрімен А. та Фрімен А. (2009). «Pro C# 2008 і платформа .NET 3.5». Apress.

4. Альбахарі, Дж., Альбахарі, Б., і Альбахарі, Б. (2015). «C# 6.0 in Nutshell: The Definitive Reference». O'Reilly Media.

5. Ліберті, Дж., Макдональд, Б. (2009). "Вивчення C# 3.0: освоїти основи C# 3.0." O'Reilly Media.

6. Petzold, C. (2010). «Програмування Microsoft Windows на C#». Microsoft Press.

7. Троельсен, А. (2012). «Pro C# 2012 і .NET 4.5 Framework». Apress.

8. Венц, Т. (2014). «Прив’язка даних за допомогою Windows Forms 2.0: Програмування додатків даних Smart Client за допомогою .NET». Аддісон-Уеслі.

9. Ліберті, Дж., Макдональд, Б. (2013). «Програмування C# 5.0: Створення Windows 8, веб-додатків і програм для робочого столу для .NET 4.5 Framework». O'Reilly Media.

10. Sells, C., & Griffiths, C. (2008). «Програмування Windows Presentation Foundation». O'Reilly Media.

11. Мейєр, Р., Нагель, К. (2014). "Прив'язка даних у Windows Forms 2.0." Аддісон-Уеслі.

12. Шарп, Дж. (2008). «Розширені Windows Forms». Apress.

13.Ліпперт, Е., Хейлсберг, А. (2012). «C# 4.0 у двох словах: повний довідник». O'Reilly Media.

14. Фрімен А. та Сендлер А. (2010). "Pro LINQ: інтегрований запит у мову C# 2010." Apress.

15. Ліберті, Дж., Макдональд, Б. (2010). «Програмування C# 4.0: Створення додатків Windows, Web і RIA для .NET 4.0 Framework». O'Reilly Media.

16. Балена, Ф. (2012). "Програмування Microsoft LINQ у Microsoft .NET Framework 4." Microsoft Press.

17. Мейєр, Р. (2011). «Кулінарна книга Windows Presentation Foundation 4.5». Packt Publishing.

18. Гріффітс К. та Селлс К. (2005). «Програмування Windows: Написання програм Windows 8 за допомогою C# і XAML». O'Reilly Media.

19. Троельсен, А. (2013). "Pro WPF 4.5 у C#: Windows Presentation Foundation у .NET 4.5." Apress.

20. Гілл, М. (2011). «Початок баз даних C# 2010: від новачка до професіонала». Apress.

ДОДАТКИ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Проректор
Українського державного
університету науки і технологій
Анатолій РАДКЕВИЧ

МОДЕЛЮВАННЯ АРХІТЕКТУРИ ПРОГРАМИ ГРАФОМ

Технічне завдання

ЛИСТ ЗАТВЕРДЖЕННЯ

44165850.01341-01-ЛЗ

Завідувач кафедри КІТ
Вадим ГОРЯЧКІН

_____

Керівник розробки
Олена КУРОП’ЯТНИК

_____

Виконавець
Владислав БАЖИН

_____

Нормоконтролер
Світлана ВОЛКОВА

_____

2023

ЗАТВЕРДЖЕНО
44165850.01341-01

МОДЕЛЮВАННЯ АРХІТЕКТУРИ ПРОГРАМИ ГРАФОМ

Технічне завдання

Листів 14

2023

АНОТАЦІЯ

Документ 44165850.94102-01 13 01 «Моделювання архітектури програми графом». Технічне завдання» входить до складу програмної документації додатку, що надає можливість визначити ефективність використання патернів за допомогою використання теорії графів.

У даному документі представлено призначення програми, область застосування програмного продукту, основні вимоги, стадії та строки розробки проекту, технічні показники, що пред’являються до програмного продукту.

ЗМІСТ

Вступ 65

1 Підстава для розробки 66

2 Призначення розробки 67

2.1 Функціональне призначення 67

2.2 Експлуатаційне призначення 67

3 Вимоги до програмного продукту 68

3.1 Вимоги до функціональних характеристик 68

3.2 Вимоги до надійності 68

3.3 Умови експлуатації 69

3.4 Вимоги до складу і параметрів технічних засобів 69

3.5 Вимоги до інформаційної і програмної сумісності 69

3.6 Вимоги до маркування і упаковки 70

3.7 Вимоги до транспортування і зберігання 70

4 Вимоги до програмної документації 71

5 Стадії та етапи розробки 72

6 Порядок контролю і приймання 74

Бібліографічний список 75

ВСТУП

Дослідження патернів прив'язки даних за зчепленням залишається актуальним та важливим напрямком у галузі розробки програмного забезпечення. Прив'язка даних – це механізм, що дозволяє автоматично синхронізувати дані між інтерфейсом користувача і бізнес-логікою програми. Зі зростанням складності та вимог до сучасних додатків стає важливим ефективне управління даними та їх відображенням. Дослідження патернів прив'язки даних за зчепленням являється актуальним та корисним напрямком, що сприяє більш ефективній та сучасній розробці програмного забезпечення.

«Моделювання архітектури програми графом» надає можливість визначити ефективність використання патернів за допомогою використання теорії графів.

1 ПІДСТАВА ДЛЯ РОЗРОБКИ

Основою для розробки є наказ в.о. ректора Українського державного університету науки і технології Пшінька О. М. «Про затвердження тем та призначення керівників дипломних проектів» №1196 ст. від 05.12. 2022 року.

Тема дипломної роботи: «Дослідження патернів прив’язки даних за зчепленням».

Керівник магістерської роботи – Куроп’ятник О. С.

2 ПРИЗНАЧЕННЯ РОЗРОБКИ

2.1 Функціональне призначення

Основною метою даного програмного додатку є визначення ефективності використання патернів прив’язки даних у окремих задачах.

2.2 Експлуатаційне призначення

Основне призначення програмного додатку полягає в автоматизації процесу перевірки впливу патерну прив’язки даних на архітектуру за допомогою теорії графів.

3 ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

3.1 Вимоги до функціональних характеристик

Вимоги до функціональних характеристик наступні:

– завантаження з файлу тексту програми;

– завантаження до програми файл коду з директорії;

– побудова графу на основі матриці суміжності, що показує зв’язки між класами;

– надання користувачеві програмного продукту висновку щодо результатів дослідження ефективності використання патернів прив’язки даних в обраному коді програми;

– лексичний аналіз тексту програм, що полягає у виділенні назв класів програм, а також їх зв’язків.

Вхідні дані:

– тексти програм на мові C#;

– назва та шлях до директорії з проєктом;

– файли з розширенням .CS.

Вихідні дані:

– матриці суміжності та граф, які відображають структурні зв’язки між класами, які визначені лексичним аналізом коду;

– текстове повідомлення з висновком щодо результатів оцінки використання патернів у даній програмі;

– текстові повідомлення, щодо успішності або неуспішності виконаних операцій роботи програми.

3.2 Вимоги до надійності

Вимоги до надійності програмного додатку наступні:

− програма повинна не допускати пошкодження даних під час своєї роботи;

− програма повинна не спричиняти втрати пам’яті;

− програма повинна стабільно працювати та кількість відмов системи не повинна перевищувати однієї відмови на 2000 запусків системи.

3.3 Умови експлуатації

Дане програмне рішення може використовуватись в умовах, відповідних до умов, які описані в документу [1].

Для забезпечення сталого функціонування програмного продукту необхідно дотримуватись таких умов:

− мінімальні навички роботи з ПК;

− програмним продуктом може користуватися людина, яка ознайомилася з керівництвом користувача;

− ЕОМ, які використовуються для роботи програмного продукту, повинні відповідати чинним вимогам та стандартам в Україні, нормативних актами з охорони праці [2];

− програмний комплекс повинен використовуватись в приміщеннях, призначених для роботи ЕОМ з наступними кліматичними умовами: температура – 21-25 °С, відносна вологість повітря 40-60 %.

3.4 Вимоги до складу і параметрів технічних засобів

Для роботи з додатком необхідний ПК, що має наступні мінімальні характеристики:

− процесор: двох’ядерний 32-, 64- або 86-бітний процесор з тактовою частотою 2 ГГц;

− оперативна пам’ять: 1 ГБ (для 32-бітних систем) або 2 ГБ (для 64-бітних систем);

− пам’ять на жорсткому диску: 500 МБ;

− периферійні пристрої: VGA або HDMI монітор з мінімальним розширенням екрану 1024x768, клавіатура, миша.

3.5 Вимоги до інформаційної і програмної сумісності

Програмний продукт має бути розроблений з використанням мови програмування C# за допомогою IDE MS Visual Studio.

Для роботи з додатком необхідно 100 Мб пам’яті для розміщення програмного додатку.

3.6 Вимоги до маркування і упаковки

У разі необхідності випуску даного програмного продукту у фізичному форматі, програмний продукт, а також електронна документація користувача повинні зберігатись на USB-носії.

Приклад маркування приведений на рис. 3.1:

Моделювання архітектури програми графом
Бажин В. А.
УДУНТ
2023

Рисунок 3.1 – Приклад маркування

3.7 Вимоги до транспортування і зберігання

Транспортування повинно забезпечувати збереження програмного продукту, його цілісність та запобігання несанкціонованого доступу до нього. Транспортування фізичної упаковки програмного продукту повинно проводитись довіреною особою та здійснюватися комплектами в упаковці з термоплівки та піни, яка захищає носій від різного виду пошкоджень. Місце зберігання програмного продукту повинне бути сухим з відсутністю пилу при температурі 21-25 °С і вологості 40-60 %, з уникненням впливу вологи та шкідників.

4 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

Програмна документація представляє собою перелік наступних документів:

– специфікація;

– текст програми;

– опис програми;

– керівництво користувача. Керівництво з визначення показників зчеплення за допомогою теорії графів.

Вся документація до програми повинна задовольняти вимогам державного стандарту до оформлення програмних документів ГОСТ 19.101-77 «Єдина система програмної документації. Види програм та програмних документів» [3].

5 СТАДІЇ ТА ЕТАПИ РОЗРОБКИ

Стадії розробки наведені в табл. 5.1.

Таблиця 5.1 – Стадії та етапи розробки

Стадії	Етапи розробки (зміст робіт)	Терміни виконання
1	2	3
1. Технічне завдання	Постановка завдання Збір початкових матеріалів Вибір і обґрунтування критеріїв ефективності і якості програми, що розробляється Обґрунтування необхідності проведення науково дослідних робіт Визначення структури вхідних і вихідних даних Попередній вибір методів рішення задач Обґрунтування доцільності застосування раніше розроблених програм Визначення вимог до технічних засобів Обґрунтування принципової можливості рішення поставленої задачі Визначення вимог до програми Визначення стадій, етапів і термінів розробки програми і документації на неї Вибір мов програмування Визначення необхідності проведення науково-дослідних робіт на подальших стадіях Узгодження і затвердження технічного завдання	01.12.23-11.12.23

Закінчення табл. 5.1

1	2	3
2. Робочий проєкт	Програмування і налагодження програми Розробка програмних документів відповідно до вимог ГОСТ 19.101-77 Розробка, узгодження і затвердження програми і методики випробувань Проведення попередніх і інших видів випробувань Коригування програми і програмної документації за наслідками випробувань	12.12.23-22.12.23
3. Впровадження	Підготовка і передача програми і програмної документації для супроводу і (або) виготовлення	23.12.23-29.12.23

6 ПОРЯДОК КОНТРОЛЮ І ПРИЙМАННЯ

Контроль за виконанням роботи виконує керівник розробки.

Прийом програмного продукту здійснюється прийомною комісією і керівником розробки.

БІБЛІОГРАФІЧНИЙ СПИСОК

1. ДСанПіН 3.3.2-007-98. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин [Текст] / Постанова Головного державного санітарного лікаря України від 10 грудня 1998 р. № 7 – К.,1998.

2. ДСН 3.3.6.042-99. Санітарні норми мікроклімату виробничих приміщень [Текст]/ Постанова Головного Державного санітарного лікаря України від 01.12.1999 № 42 – К., 1999.

3. ГОСТ 19.101-77. Види програм та програмних документів [Текст]/ Постанова Державного комітету стандартів Ради Міністрів СРСР від 20 травня 1977 г. – М., 1977.

4. Еріх Гамма, Річард Хелм, Ральф Джонсон, Джон Вліссідес. "Design Patterns: Elements of Reusable Object-Oriented Software " (Прийоми об'єктно-орієнтованого проектування).

5. Ендрю Троелсен. "Pro C# 7: With .NET and .NET Core" (Програмування на C# 7 з використанням .NET і .NET Core).

6. Брайан Нойс. "Data Binding with Windows Forms 2.0: Programming Smart Client Data Applications with .NET".

7. Якуб Гіжка. "Angular Development with TypeScript" (Розробка на Angular з використанням TypeScript).

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Проректор
Українського державного
університету науки і технологій
Анатолій РАДКЕВИЧ

МОДЕЛЮВАННЯ АРХІТЕКТУРИ ПРОГРАМИ ГРАФОМ

Робочий проєкт

ЛИСТ ЗАТВЕРДЖЕННЯ

44165850.01341-01-ЛЗ

Завідувач кафедри КІТ
Вадим ГОРЯЧКІН

_____

Керівник розробки
Олена КУРОП’ЯТНИК

_____

Виконавець
Владислав БАЖИН

_____

Нормоконтролер
Світлана ВОЛКОВА

_____

2023

ЗАТВЕРДЖЕНО
44165850.01341-01

МОДЕЛЮВАННЯ АРХІТЕКТУРИ ПРОГРАМИ ГРАФОМ

Специфікація

Листів 2

2023

Позначення	Найменування	Примітка
44165850.01341-01-ЛЗ	Лист затвердження	
44165850.01341-01 12 01-ЛЗ	Лист затвердження	
44165850.01341-01 12 01	Текст програми	
44165850.01341-01 13 01-ЛЗ	Лист затвердження	
44165850.01341-01 13 01	Опис програми	
44165850.01341-01 І3 01-ЛЗ	Лист Затвердження	
44165850.01341-01 І3 01	Керівництво користувача. Керівництво з визначення ефективності використання патернів прив’язки даних за зчепленням	

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖЕНО
44165850.01341-01 13 01

МОДЕЛЮВАННЯ АРХІТЕКТУРИ ПРОГРАМИ ГРАФОМ

Опис програми

Листів 15

2023

АНОТАЦІЯ

Документ 44165850.01341-01 13 01 ««Моделювання архітектури програми графом». Опис програми» входить до складу програмної документації додатку, який надає можливість визначати ефективність патернів прив’язки даних за зчепленням за допомогою порівняння двох кодів програм.

У даному документі представлено призначення програми, область застосування програмного продукту, основні вимоги, стадії та строки розробки проекту, технічні показники, що пред’являються до програмного продукту.

ЗМІСТ

1 Загальні відомості 84

2 Функціональне призначення 85

3 Опис логічної структури 86

3.1 Алгоритм та методи програми 86

3.2 Структура програми з описом функцій складових частин і зв’язки між ними 86

3.3 Зв’язки програми з іншими програмами 87

4 Використані технічні засоби 88

5 Виклик та завантаження 89

6 Вхідні дані 90

7 Вихідні дані 91

8 Опис інтерфейсу користувача 92

8.1 Опис станів програми 92

8.2 Опис переходів між станами програми 92

8.3 Опис керування діалогом 92

8.4 Форматування екрана 93

9 Порядок роботи з програмою 94

10 Повідомлення 95

1 ЗАГАЛЬНІ ВІДОМОСТІ

Програмний додаток «Моделювання архітектури програми графом» являє собою інструментарій для визначення ефективності використання патернів прив’язки даних в конкретних задачах.

Для коректного функціонування програми необхідно щоб на персональному комп’ютері було встановлено операційну систему Windows 7, або вище.

Програма реалізована на мові програмування C# з використанням технології .NET в середовищі MS Visual Studio 2022.

2 ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Метою програмного додатку є визначення ефективності використання патернів прив’язки даних за зчепленням у програмі, що написана мовою програмування C#.

Функціональні вимоги до додатку:

− лексичний аналіз кодів програм;

− побудова двох графів, що представляють собою візуалізацію архітектури обох програм;

− порівняння між собою цих програм за допомогою отриманого графу, а також окремо визначених критеріїв.

Відповідно до функціональних вимог додатку було визначено наступні вимоги до даних програми:

− програмний код програм, який досліджується, має бути на мові програмування C#;

− файли, що містять програмний код, повинні мати розширення .cs.

3 ОПИС ЛОГІЧНОЇ СТРУКТУРИ

3.1 Алгоритм та методи програми

Програмний додаток складається з таких логічних блоків:

− лексичний аналіз кодів програм, виділення назв класів та зв’язків між ними;

− побудова орієнтованих графів на основі лексичного аналізу;

− аналіз графів за допомогою розробленого методу, а також надання висновку щодо ефективності використання патернів прив’язки даних.

Лексичний аналіз складається з наступних етапів:

− видалення одно строкових та багато строкових коментарів, строкових літералів, пробілів, знаків табуляції та ін.;

− виділення назв класів та їх зв’язків;

− аналіз зв’язків між класами, що передбачає визначення кількості звертань, а також уточнення, який саме клас до якого звертається.

На основі лексичного аналізу будуються графи, де вершини – представляють класи, а ребра – зв’язки між цими класами.

3.2 Структура програми з описом функцій складових частин і зв’язки між ними

Рівень логіки даної програми включає в себе такі класи:

- Graph – відповідає за побудову графа;
- LexicalAnalysis – відповідає за лексичний аналіз коду;
- ClassConnections – відповідає за зв’язки між класами у коді;
- Preliminary – відповідає за попередню обробку коду;
- Efficiency – має методи, що визначають ефективність патернів;

Рівень представлення включає в себе наступні класи:

- MainForm – відповідає за головну форму програми;
- ResultForm – відповідає за додаткову форму з результатами визначення ефективності використання патернів;

Рівень даних включає в себе один клас InputData, який відповідає за методи, що завантажують вхідні дані.

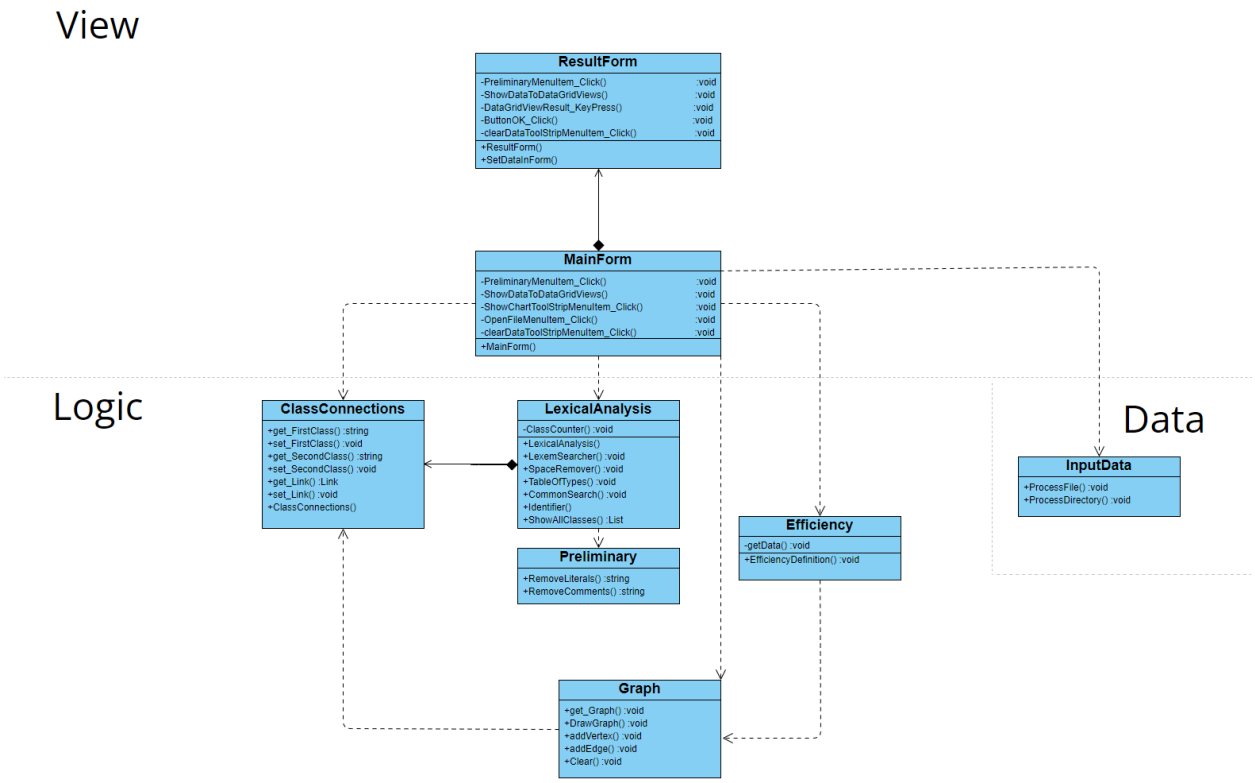

Рисунок 3.1 – Діаграма класів

3.3 Зв’язки програми з іншими програмами

Для коректної роботи програмного додатку необхідні такі програми:

− операційна система Windows 7, або більше;

− framework .NET 6.0, або більше.

4 ВИКОРИСТАНІ ТЕХНІЧНІ ЗАСОБИ

Для коректного функціонування програмного забезпечення достатньо мати робочий персональний комп’ютер, що має наступні технічні характеристики:

– процесор з тактовою частотою 1 Ггц;

– не менше 512 Мб ОЗУ;

– 150 Мб вільного місця на жорсткому диску;

– архітектура x86 або x64;

– периферійні пристрої – VGA-монітор з мінімальним розширенням екрану 1024x768, клавіатура, миша.

5 ВИКЛИК ТА ЗАВАНТАЖЕННЯ

Програма викликається шляхом подвійного натискання лівою кнопкою миші на виконавчий файл програми «Graphs.exe», що знаходиться у папці де було встановлено програмний додаток.

Файли для завантаження у програмний додаток, а саме файли з розширенням .cs, що містять програмний код можуть зберігатися у будь-якому місці, що обере користувач.

6 ВХІДНІ ДАНІ

Вхідні дані програмного продукту:

− тексти програм на мові C#;

− назва та шлях до директорії зі збереженим файлом;

− файли з розширенням .cs.

7 ВИХІДНІ ДАНІ

Вихідні дані програмного продукту:

− графи, які моделюють архітектуру програм;

− окрема форма з висновками, щодо ефективності використання патернів;

− текстові повідомлення, щодо успішності або неуспішності виконаних операцій.

8 ОПИС ІНТЕРФЕЙСУ КОРИСТУВАЧА

8.1 Опис станів програми

Стани програми наведено у табл. 8.1.

Таблиця 8.1 – Стани програми

№ стану	Назва стану	Опис стану	Рекомендовані дії
1	Запуск програми	Програма запускається	Очікування головного вікна програми
2	Завантаження файлу з програмним кодом на мові C#	Вибір та завантаження файлів з розширенням .cs	Обрати необхідний файл з розширенням .cs
3	Введення програмного коду на мові C#	Введення програмного коду на мові C# у віповідне текстове вікно	Ввести програмний код на мові C#
4	Побудова графів	Лексичний аналіз кодів та побудова графів	Натиснути кнопку «Побудувати граф»
5	Визначення ефективності використання патернів	Визначення ефективності використання патернів на основі розробленого методу	Натиснути кнопку «Висновки»

8.2 Опис переходів між станами програми

Схема переходів наведено на рис. 8.1. Цифри є номером стану програми (табл. 8.1). Вийти з програми можливо під час усіх станів крім першого.

8.3 Опис керування діалогом

Керування діалогом відбувається за допомогою екранних форм та головного меню додатку.

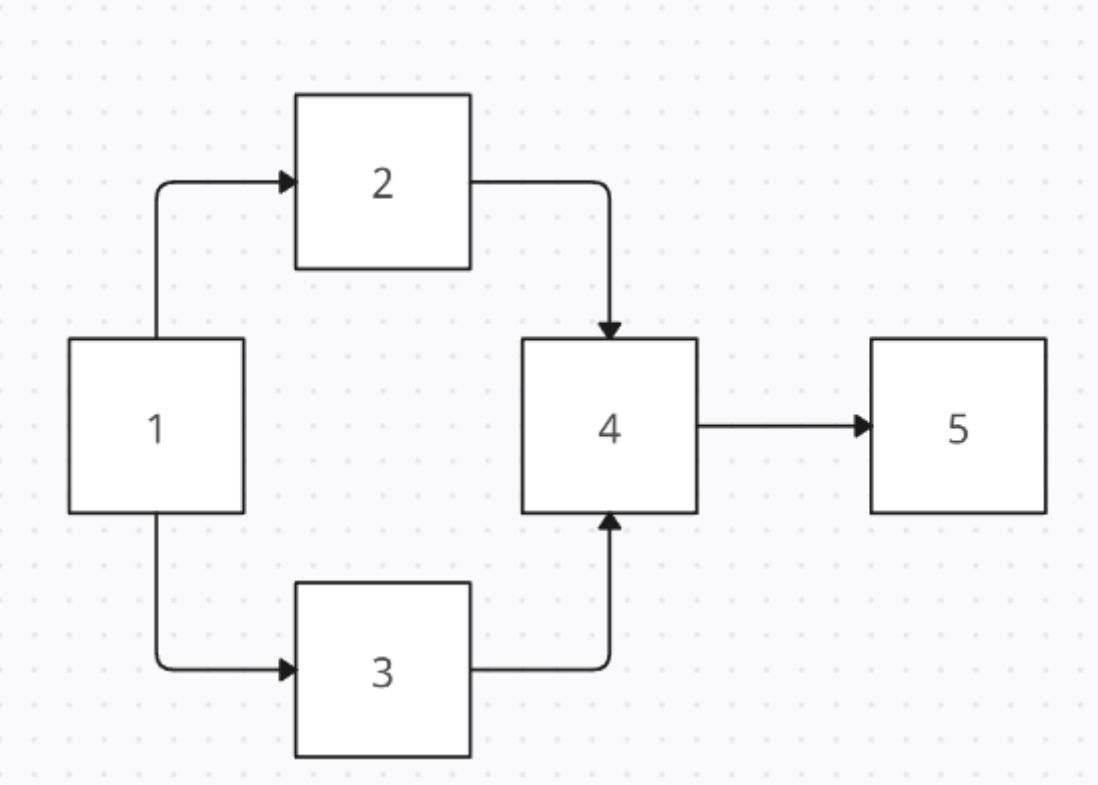

Рисунок 8.1 – Схема переходів між станами програми

8.4 Форматування екрана

Засоби форматування не передбачені. Розміри підібрані оптимально, щоб можна було зручно користуватися у будь-якому розширенні монітору.

9 ПОРЯДОК РОБОТИ З ПРОГРАМОЮ

Порядок роботи з програмою передбачає наступну послідовність операцій:

− запуск додатку за допомогою файлу «Graphs.exe»;

− введення або завантаження кодів програм або проєктів на мові C#;

− побудова графів, що моделюватимуть архітектуру програм;

− визначення ефективності використання патернів прив’язки даних.

10 ПОВІДОМЛЕННЯ

У табл. 10.1 наведені повідомлення, що можуть з’явитися під час роботи з програмою у користувача.

Таблиця 10.1 – Список повідомлень користувачу

Текст повіодмлення	Опис ситуації	Рекомендовані дії
Обраний файл не має розирешння .cs	Обрано файл з невірним розширенням	Обрати коректний файл з розширенням .cs
Програмний код не введено	Програмний код відсутній у текстовому полі	Ввести програмний код вручну, або завантажити файл .cs
Не вдалося побудувати графи	Програмний код не представляє можливим побудувати графи	Перевірити, чи містить програмний код класи, що мають зв’язки

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖЕНО
44165850.01341-01 І3 01

МОДЕЛЮВАННЯ АРХІТЕКТУРИ ПРОГРАМИ ГРАФОМ

Керівництво користувача. Керівництво з визначення ефективності використання патернів прив’язки даних за зчепленням

Листів 12

2023

АНОТАЦІЯ

Документ 44165850.01341-01 І3 01 ««Моделювання архітектури програми графом». Керівництво користувача. Керівництво з визначення ефективності використання патернів прив’язки даних за зчепленням» входить до складу програмної документації додатку, який надає можливість визначати ефективність використання патернів в конкретних задачах.

У даному документі представлено призначення та умови застосування програмного продукту, інструкції з підготовки до роботи, а також опис основних операцій та аварійних ситуацій в програмному додатку.

ЗМІСТ

Вступ 99

1 Призначення та умови застосування 100

1.1 Функціональне призначення програми 100

1.2 Вимоги до складу і параметрів технічних засобів 100

1.3 Вимоги до інформаційної і програмної сумісності 100

2 Підготовка до роботи 101

3 Опис операцій 102

4 Аварійні ситуації 103

5 Рекомендації щодо засвоєння 104

ВСТУП

Програмний додаток «Моделювання архітектури програми графом» використовується користувачами, що пишуть код з використанням патернів, а також студентами, що їх вивчають.

Програмний додаток надає можливість визначити ефективність використання патернів прив’язки даних за зчепленням у програмах, що написані на мові програмування C#.

Для використання програми необхідні мінімальні навички роботи з ПК та операційною системою Windows 7, або вище.

Для коректної роботи з програмним додатком необхідно ознайомитися з описом програми та керівництвом користувача.

1 ПРИЗНАЧЕННЯ ТА УМОВИ ЗАСТОСУВАННЯ

1.1 Функціональне призначення програми

Основною метою даного програмного додатку є визначення ефективності використання патернів прив’язки даних на основі моделювання двох програм графом.

До функціональних можливостей програмного додатку входить:

− лексичний аналіз програмного коду на мові C# та побудова графів, що моделюють архітектуру програм;

− аналіз отриманих графів на основі розробленого методу за визначеними критеріями;

− надання висновку у вигляді додаткової форми, де детально описано вплив патернів на архітектуру програми.

1.2 Вимоги до складу і параметрів технічних засобів

Для коректного функціонування програмного забезпечення достатньо мати робочий персональний комп’ютер, що має наступні технічні характеристики:

– процесор з тактовою частотою 1 Ггц;

– не менше 512 Мб ОЗУ;

– 150 Мб вільного місця на жорсткому диску;

– архітектура x86 або x64;

– периферійні пристрої – VGA-монітор з мінімальним розширенням екрану 1024x768, клавіатура, миша.

1.3 Вимоги до інформаційної і програмної сумісності

Для коректного функціонування програми необхідно щоб на персональному комп’ютері було встановлено операційну систему Windows 7 або вище, Framework .NET 6.

2 ПІДГОТОВКА ДО РОБОТИ

Для виклику програми двічі натисніть лівою кнопкою миші по іконці виконавчого файлу програми «Graphs.exe», що знаходиться у папці де було розміщено програмний додаток. Для роботи з програмним додатком завантажте підготовлені файли з програмним кодом на мові C# з розширенням .cs до віподвіного поля на головній формі.

3 ОПИС ОПЕРАЦІЙ

Після старту роботи з програмним додатком можливі наступні операції:

1. Завантаження програмного коду на мові C#, що міститься в одному файлі: завантажте файл з розширенням .cs за допомогою кнопки «Завантажити код», або завантажте вже збережений код з побудованими графами через меню «Файл» та «Завантажити готовий граф»;

2. Така сама процедура з попереднього пункту для другого файлу;

3. Введення програмного коду на мові C#: введіть програмний код у відповідне текстове поле;

4. Побудова графу: натисніть кнопку «Побудувати граф для першого коду» та «Побудувати граф для другого коду»;

5. Визначення ефективності патернів: натисніть кнопку «Висновки», щоб відкрити додаткову форму з описом впливу патернів на архітектуру;

6. Перегляд довідки: оберіть пункт головного меню «Про програму», або «Інструкція з користування».

4 АВАРІЙНІ СИТУАЦІЇ

У разі виявлення помилок у даних програми, необхідно натиснути кнопку «Очистити форму» та зробити все ще раз, або перезапустити додаток та зробити все з початку.

У разі виявлення інших аварійних ситуацій, необхідно закрити програмний додаток та зв’язатися з персоналом, що супроводжує даний додаток.

5 РЕКОМЕНДАЦІЇ ЩОДО ЗАСВОЄННЯ

Інтерфейс програмного додатку зображено на рис. 5.1.

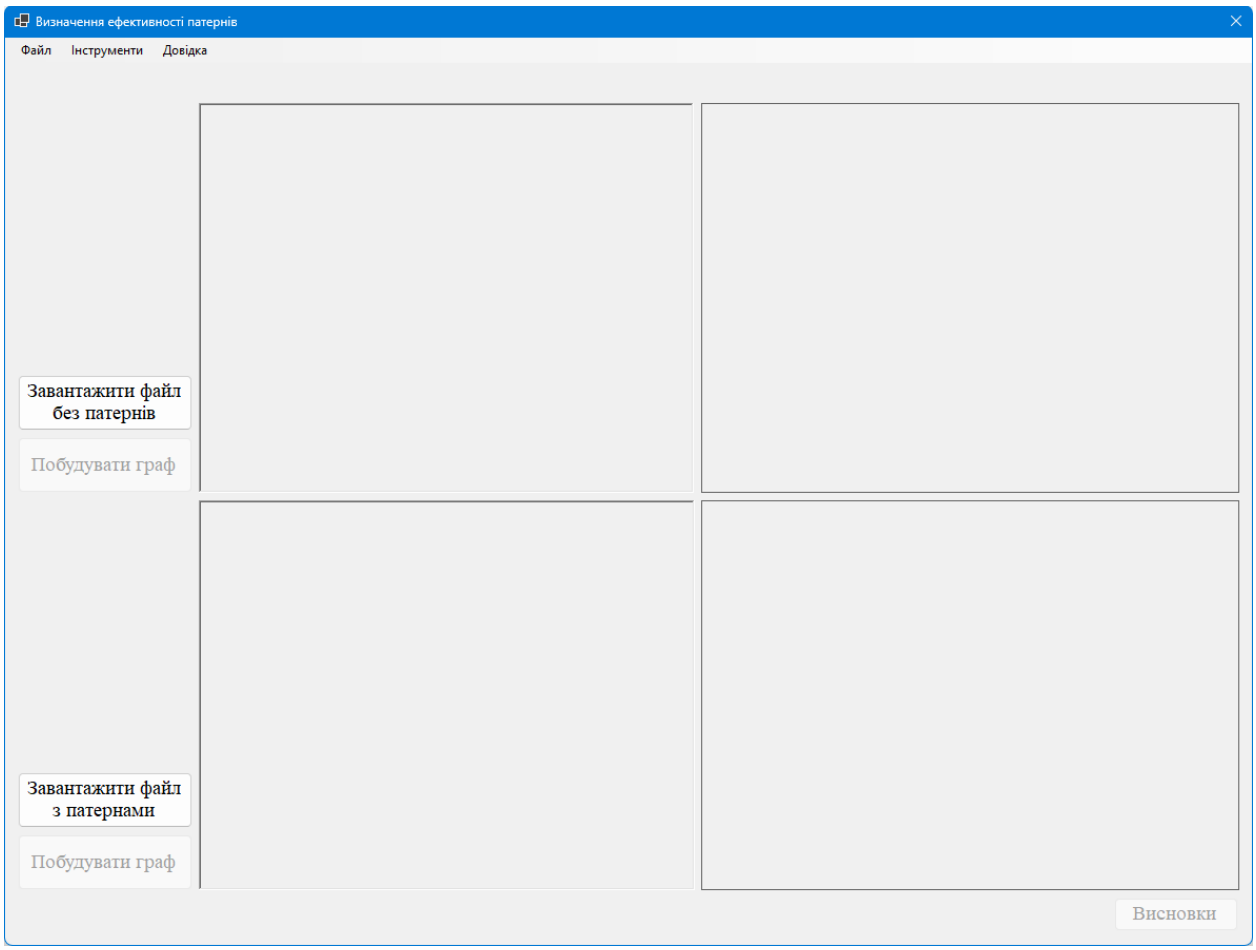

Рисунок 5.1 – Головна форма програми

Для завантаження програмного коду на мові C#, натисніть кнопку «Завантажити файл без патернів» та відкрийте відповідний файл з розширенням .cs. Після цього у текстовому полі справа від кнопки з’явиться текст Вашої програми (рис. 5.2).

Програмний код також можна ввести вручну. Для цього треба поставити курсор комп’ютерної миші у текстовому полі та почати вводити текст програми на мові C# вручну.

Для другого коду зробіть такий самий алгоритм, натиснувши кнопку «Завантажити файл з патернами» та відкрити відповідний файл з розширенням .cs (рис. 5.3).

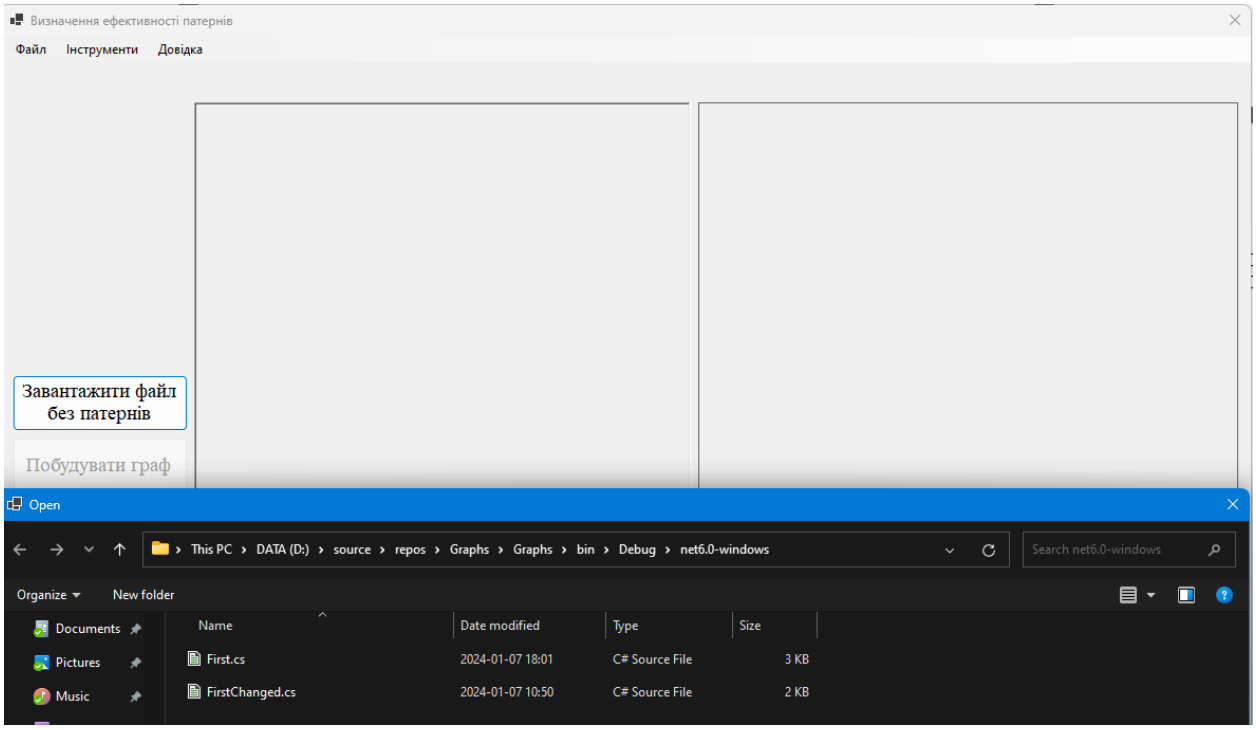

Рисунок 5.2 – Завантаження програмного коду

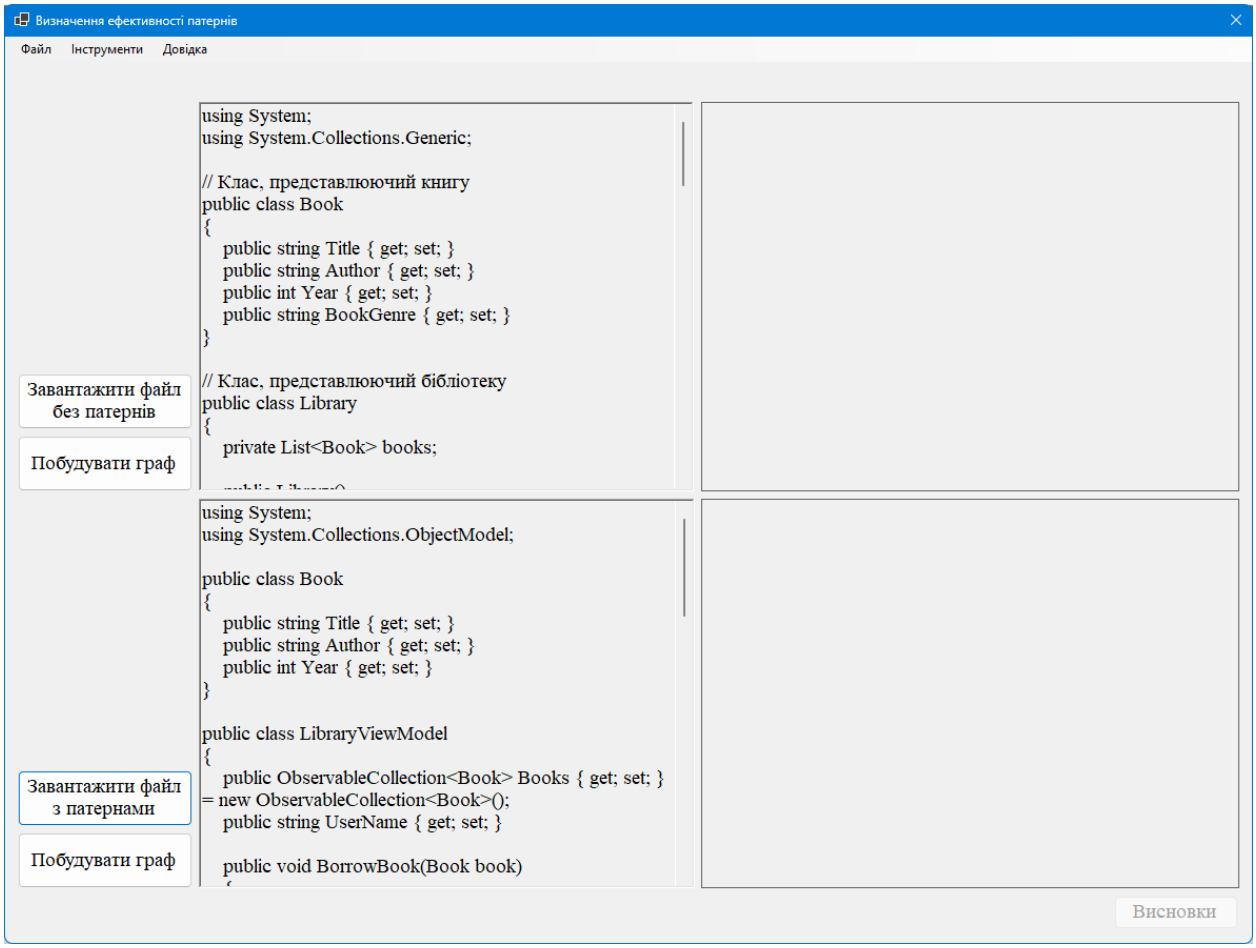

Рисунок 5.3 – Завантажено обидва програмних коди

Щоб побудувати графи натисніть кнопку «Побудувати граф», яка стане доступною після успішного завантаження програмного коду. Верхня кнопка будує граф для першого коду. Нижня кнопка будує граф для другого коду (рис. 5.4).

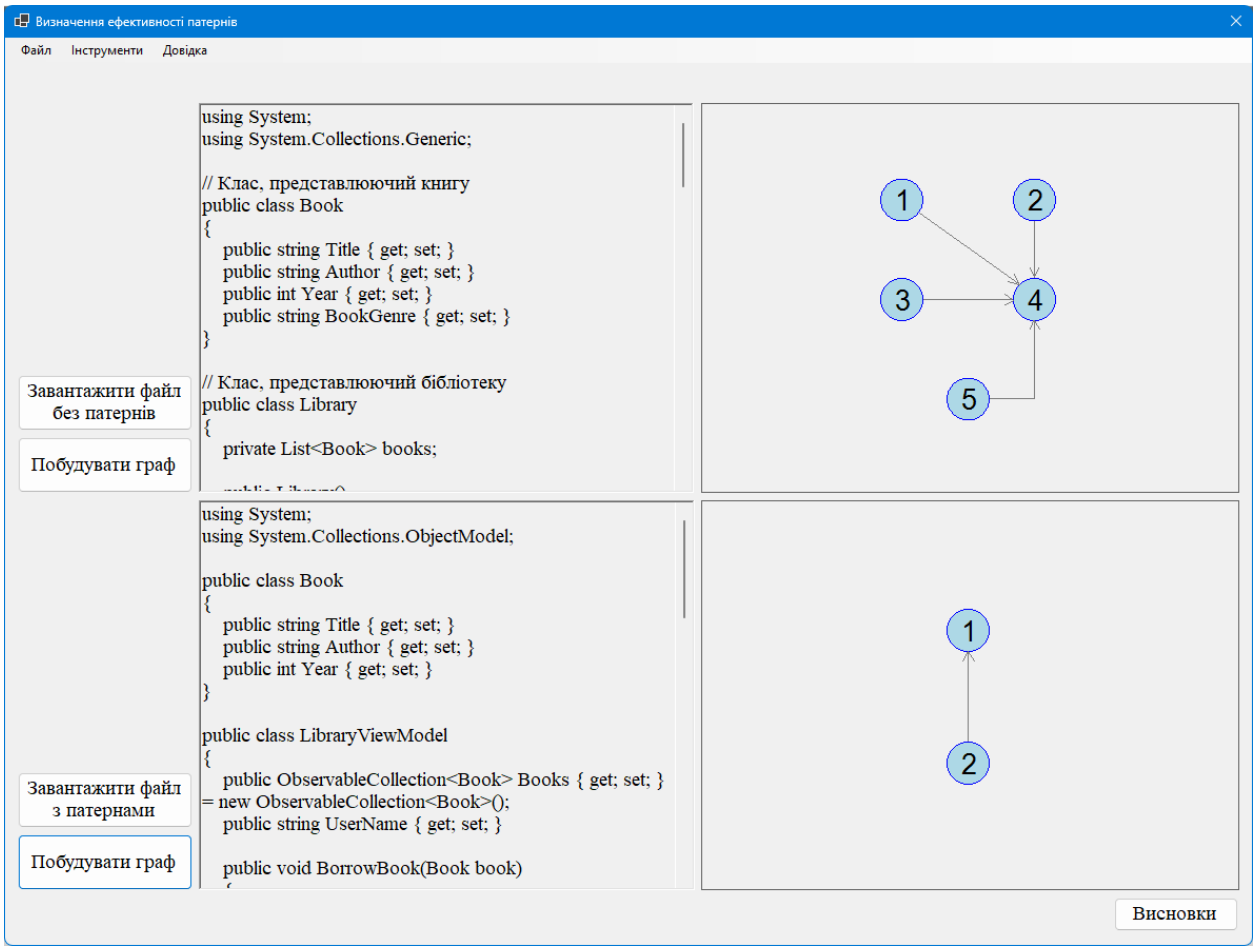

Рисунок 5.4 – Побудовані графи

Для отримання висновків щодо результатів визначення ефективності натисніть кнопку «Висновки», яка стає доступною після успішної побудови двох графів. У висновках надано інформацію щодо кількості класів, зв’язків між класами, а також їх порівняння. Перші дві таблиці – окрема інформація про кожний граф. Третя таблиця – їх порівняння. Зелений колір свідчить про те, що патерни позитивно вплинули на архітектуру (рис. 5.5.). Червоний колір – навпаки.

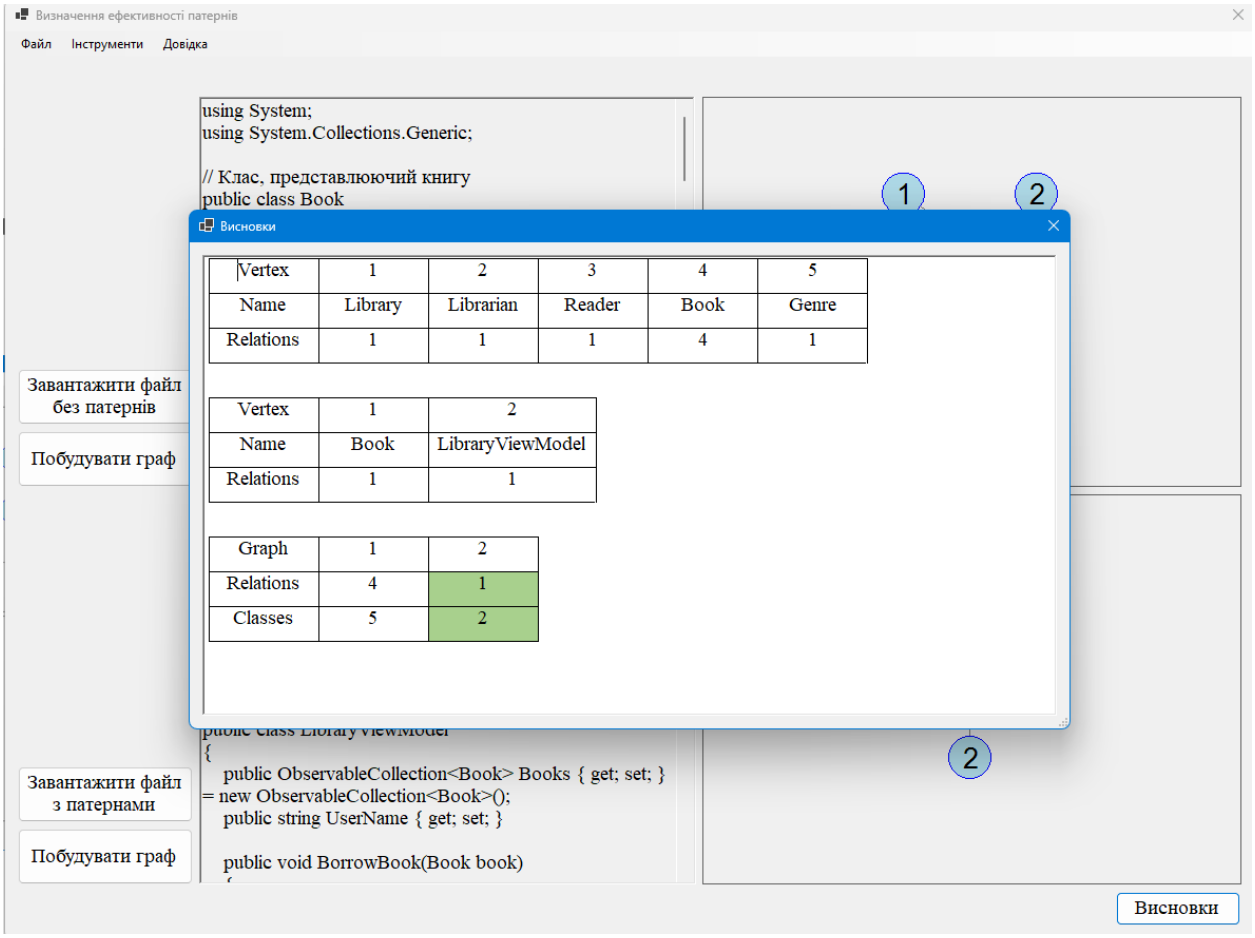

Рисунок 5.5 – Форма з висновками

Також на головній формі програми є меню. В ньому є «Файл», натиснувши який, надається можливість зберегти отримані результати у форматі .png. «Інструменти» відкриває меню з можливістю повністю очистити головну форму додатку. «Довідка» відкриває меню, в якому можна натиснути «Прочитати про програму», де описано суть роботи. А також «Інстркція з користування», де розписаний алгоритм роботи з програмою (рис. 5.6).

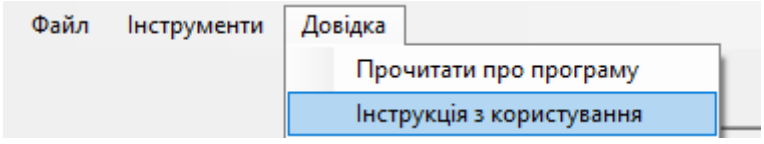

Рисунок 5.6 – Меню головної форми

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖЕНО
44165850.01341-01 12 01

МОДЕЛЮВАННЯ АРХІТЕКТУРИ ПРОГРАМИ ГРАФОМ

Текст програми

Листів 1

2023

АНОТАЦІЯ

Документ 44165850.01341-01 12 01 ««Моделювання архітектури програми графом». Текст програми» входить до складу програмної документації додатку, який надає можливість визначати ефективність використання патернів прив’язки даних. В документі міститься текст програми. Програма реалізована на мові C# у програмному середовищі MS Visual Studio.

ЗМІСТ

1 Структура програми 111

2 Текст програми 112

1 СТРУКТУРА ПРОГРАМИ

Проєкт програмного додатку складається з трьох рівнів: логіки, представлення та даних. Рівень логіки відповідає за лексичний аналіз програмних кодів, побудову графів на основі цих аналізів, формування висновків щодо ефективності використання патернів прив’язки даних. Рівень даних відповідає за методи, що завантажують вхідні дані.

Опис кожного файлу структури:

- Graph.cs – файл рівня логіки, що відповідає за побудову графа;
- LexicalAnalysis.cs – файл рівня логіки, що відповідає за лексичний аналіз коду;
- ClassConnections.cs – файл рівня логіки, що відповідає за зв’язки між класами у коді;
- Preliminary.cs – файл рівня логіки, що відповідає за попередню обробку коду;
- Efficiency.cs – файл рівня логіки, що має методи, які визначають ефективність патернів;
- MainForm.cs – файл рівня представлення, що відповідає за головну форму програми;
- ResultForm.cs – файл рівня представлення, що відповідає за додаткову форму з результатами визначення ефективності використання патернів;
- InputData.cs – файл рівня даних, який відповідає за методи, що завантажують вхідні дані.

2 ТЕКСТ ПРОГРАМИ

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.IO;
using System.Text;
using System.Windows.Forms;

namespace Graph
{
  public sealed partial class FormMain : Form
  {
    private const string FileName = "saved.txt";
    private const string RtfFileName = "doc.rtf";
    private const int RandomFactor = 3;
    private readonly string _fullFileName = Path.Combine(Path.GetDirectoryName(Application.ExecutablePath), FileName);
    private byte _graphSize = 1;
    private readonly GraphPainter _graphPainter;
    private byte[,] _matrix;
    private Timer _timer;
    private Queue<int> _selected;

    public FormMain()
    {
      InitializeComponent();
      _graphPainter = new GraphPainter(pictureBox.CreateGraphics(), BackColor);
      dataGridView.RowCount = _graphSize;
      buttonProcess.Enabled = _graphPainter.IsDrawing;

    }

    private void trackBarSize_ValueChanged(object sender, EventArgs e)
    {
      dataGridView.SuspendLayout();
      _graphSize = (byte) ((TrackBar) sender).Value;
      dataGridView.ColumnCount = _graphSize;
      dataGridView.RowCount = _graphSize;
      for (var i = 0; i < _graphSize; i++ë)
      {
        for (var j = i; j < _graphSize; j++)
        {
          var cell = dataGridView[i, j];
          cell.Value = null;
          if (i == j) cell.Style.BackColor = Color.Gray;
        }
      }
      dataGridView.ResumeLayout();
      _matrix = null;
      buttonDraw.Enabled = buttonProcess.Enabled = buttonSave.Enabled = _matrix != null;
      _graphPainter.Clear();
      buttonProcess.Enabled = false;

    }
```

```
        private void buttonRandom_Click(object sender, EventArgs e)
        {
            AddResult("Побудова графу за вказаним кодом");
            var random = new Random();
            for (var i = 0; i < _graphSize; i++)
            {
                for (var j = i + 1; j < _graphSize; j++)
                {
                    dataGridView[i, j].Value = random.Next(0, 10) < RandomFactor ? 1 : 0;
                    //AddResult($"Добавлено ребро графа: ({i + 1}; {j + 1}) весом {dataGridView[i, j].Value}");
                }
            }
            _matrix = CreateMatrix(dataGridView);
            buttonDraw.Enabled = buttonProcess.Enabled = buttonSave.Enabled = _matrix != null;
            _graphPainter.Clear();
            buttonProcess.Enabled = _graphPainter.IsDrawing;

        }

        private void dataGridView_CellValueChanged(object sender, DataGridViewCellEventArgs e)
        {
            var grid = sender as DataGridView;
            if (grid != null) grid[e.RowIndex, e.ColumnIndex].Value = grid[e.ColumnIndex, e.RowIndex].Value;
        }

        private void AddResult(string value)
        {
            listBoxResult.Items.Add(value);
            var visibleItems = listBoxResult.ClientSize.Height / listBoxResult.ItemHeight;
            listBoxResult.TopIndex = Math.Max(listBoxResult.Items.Count - visibleItems + 1, 0);
        }

        private static byte[,] CreateMatrix(DataGridView grid)
        {
            var size = grid.RowCount;
            var matrix = new byte[size, size];
            for (short i = 0; i < size; i++)
            for (short j = 0; j < size; j++)
                matrix[i, j] = i == j ? (byte) 0 : Convert.ToByte(grid[j, i].Value);
            return matrix;
        }

        private void buttonSave_Click(object sender, EventArgs e)
        {
            var gs = _graphSize;
            var sb = new StringBuilder();
            sb.AppendLine($"{gs}");
            if (gs > 1)
            {
                var dg = dataGridView;
                for (var i = 0; i < gs - 1; i++)
                {
                    for (var j = i + 1; j < gs; j++)
                    {
                        sb.Append(dg[i, j].Value);
```

```
                if (j < gs - 1) sb.Append(";");
              }
              if (i < gs - 2) sb.AppendLine();
            }
          }
          File.WriteAllText(_fullFileName,
sb.ToString());
        }

        private void buttonLoad_Click(object
sender, EventArgs e)
        {
          try
          {
            var strs =
File.ReadAllLines(_fullFileName);
            trackBarSize.Value =
int.Parse(strs[0]);
            var gs = _graphSize;
            if (gs <= 1) return;
            var dg = dataGridView;
            AddResult("Заполнение графа
сохраненными значениями");
            for (var i = 0; i < gs - 1; i++)
            {
              var vals = strs[i + 1].Split(';');
              var index = 0;
              for (var j = i + 1; j < gs; j++)
              {
                dg[i, j].Value = vals[index];
                index++;
                //AddResult($"Добавлено
ребро графа: ({i + 1}; {j + 1}) весом {dg[i,
j].Value}");
              }
            }
            _matrix = CreateMatrix(dg);
            buttonDraw.Enabled =
buttonProcess.Enabled = buttonSave.Enabled =
_matrix != null;
            _graphPainter.Clear();
          }
          catch (Exception ex)
          {
            AddResult(ex.Message);
          }
          buttonProcess.Enabled =
_graphPainter.IsDrawing;

        }

        private void buttonProcess_Click(object
sender, EventArgs e)
        {
          buttonSave.Enabled = false;
          buttonLoad.Enabled = false;
          buttonDraw.Enabled = false;
          buttonRandom.Enabled = false;
          buttonProcess.Enabled = false;
          trackBarSize.Enabled = false;
          Process(_matrix);
        }

        private void buttonDraw_Click(object
sender, EventArgs e)
        {
          if (_matrix == null) return;
          _graphPainter.DrawGraph(_matrix);
          buttonProcess.Enabled =
_graphPainter.IsDrawing;
        }

        private void Process(byte[,] matrix)
        {
          _selected = new Queue<int>();
```

```
            var queueVertex = new Queue<int>();
// Очередь вершин на рассмотрение
            var visitedVertex = new List<int>(); //
Список уже рассмотренных вершин

            queueVertex.Enqueue(0); // Начинаем
обход с 1й вершины
            while (queueVertex.Count != 0) //До
тех пор, пока не обошли все вершины
            {
              var index = queueVertex.Dequeue();
// Выталкиваем из начала списка индекс
текущей вершины
              for (var j = 0; j < _graphSize; j++)
                if (!matrix[index, j].Equals(0) &&
!visitedVertex.Contains(j) &&
!queueVertex.Contains(j))
                  // Если ребно не нулевое и
вершина еще не была рассмотрена и не
находится в очереди на рассмотрение
                  queueVertex.Enqueue(j);      //
Добавить вершину в список на рассмотрение
              _selected.Enqueue(index);
              visitedVertex.Add(index);
            }
            _timer = new Timer {Interval = 1000};
            _timer.Tick += timer_Tick;
            _timer.Start();
          }

        private void timer_Tick(object sender,
EventArgs e)
    using System;
    using System.Collections.Generic;
    using System.Drawing;
    using System.Text.RegularExpressions;

    namespace Graph
```

```
        {
          if (_selected.Count > 0)
          {
            var node = _selected.Dequeue();

_graphPainter.DrawSelectedNode(node);
            AddResult($"-->{node + 1}");
          }
          else
          {
            buttonSave.Enabled = true;
            buttonLoad.Enabled = true;
            buttonDraw.Enabled = true;
            buttonRandom.Enabled = true;
            trackBarSize.Enabled = true;
            buttonProcess.Enabled =
_graphPainter.IsDrawing;
            _timer.Stop();
          }
        }

        private void buttonDoc_Click(object
sender, EventArgs e)
        {
          var form = new
FormDoc(Path.Combine(Path.GetDirectoryName(
Application.ExecutablePath), RtfFileName));
          form.ShowDialog(this);
        }
      }
    }

    {
      internal sealed class GraphPainter
      {
        private readonly Graphics _graphics;
        private readonly Color
_backgroundColor;
```

```
        private PointF[] _nodes;
        private bool[] _selected;
        private const float NodeRadius = 20;
        private readonly Font _font = new Font("Arial", 20);
        private byte[,] _matrix;
        public bool IsDrawing { get; private set; }

        public GraphPainter(Graphics graphics, Color backgroundColor)
        {
            _graphics = graphics;
            _backgroundColor = backgroundColor;
        }

        public void Clear()
        {
            IsDrawing = false;
            _graphics.Clear(_backgroundColor);
        }

        public void DrawGraph(byte[,] matrix)
        {
            Clear();
            _matrix = matrix;
            _nodes = CalculateNodes(matrix);
            _selected = new bool[_nodes.GetLength(0)];
            var len = matrix.GetLength(0);
            for (var i = 0; i < len; i++)
            {
                for (var j = i + 1; j < len; j++)
                {
                    if (matrix[i, j] == 1)
                    {
                        DrawEdge(_nodes[i], _nodes[j]);
                    }
                }
            }
            for (var i = 0; i < _nodes.Length; i++)
            {
                DrawNode(i);
            }
            IsDrawing = true;
        }

        private PointF[] CalculateNodes(byte[,] matrix)
        {
            var len = matrix.GetLength(0);
            var cols = (int)Math.Round(Math.Sqrt(len), 0);
            var rows = len % cols == 0 ? len / cols : (len / cols) + 1;
            var rowFactor = _graphics.VisibleClipBounds.Height / (rows + 1);
            var colFactor = _graphics.VisibleClipBounds.Width / (cols + 1);
            var result = new PointF[len];
            var index = 0;
            for (var r = 1; r <= rows; r++)
            {
                if (r == rows)
                {
                    colFactor = _graphics.VisibleClipBounds.Width / (len - index + 1);
                }
                for (var c = 1; c <= cols; c++)
                {
                    if (index >= len) break;
```

```
                result[index] = new PointF(c * colFactor, r * rowFactor);
                index++;
            }
        }
        return result;
    }

    private void DrawNode(int nodeIndex, bool selected = false)
    {
        var point = _nodes[nodeIndex];
        var rect = new RectangleF(point.X - NodeRadius, point.Y - NodeRadius, NodeRadius * 2, NodeRadius * 2);
        _graphics.FillEllipse(selected ? Brushes.Yellow : Brushes.LightBlue, rect);
        _graphics.DrawEllipse(selected ? Pens.DarkRed : Pens.Blue, rect);
        _graphics.DrawString($"{nodeIndex + 1}", _font, Brushes.Black, point.X - NodeRadius / (nodeIndex < 9 ? 1.9F : 1.05F), point.Y - NodeRadius / 1.3F);
    }

private void analyzeButton_Click(object sender, EventArgs e)
    {
        string sourceCode = codeRichTextBox.Text;

        List<string> classNames = ExtractClassNames(sourceCode);

        // Выводим имена классов в listBox
        classListBox.Items.Clear();
        foreach (var className in classNames)
        {
            classListBox.Items.Add(className);
        }
    }

    private List<string> ExtractClassNames(string sourceCode)
    {
        List<string> classNames = new List<string>();
```

```
    private void DrawEdge(PointF point1, PointF point2, bool selected = false)
    {
        _graphics.DrawLine(new Pen(selected ? Color.DarkOrange : Color.Gray, 2F), point1, point2);
    }

    public void DrawSelectedNode(int index)
    {
        _selected[index] = true;
        for (var i = 0; i < _selected.Length; i++)
        {
            if (_matrix[index, i] != 1 || !_selected[i]) continue;
            DrawEdge(_nodes[i], _nodes[index], true);
            DrawNode(i, true);
        }
        DrawNode(index, true);
    }
}
}
```

```
            // Используем регулярное выражение для поиска ключевого слова "class" и за ним следующего идентификатора
            Regex classRegex = new Regex(@"\bclass\s+(\w+)\b");

            // Находим все совпадения
            MatchCollection matches = classRegex.Matches(sourceCode);

            // Извлекаем имена классов из совпадений
        namespace Graph
        {
          partial class FormDoc
          {
            /// <summary>
            /// Required designer variable.
            /// </summary>
            private System.ComponentModel.IContainer components = null;

            /// <summary>
            /// Clean up any resources being used.
            /// </summary>
            /// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
            protected override void Dispose(bool disposing)
            {
              if (disposing && (components != null))
              {
                components.Dispose();
              }
              base.Dispose(disposing);
            foreach (Match match in matches)
            {
              if (match.Groups.Count > 1)
              {
                string className = match.Groups[1].Value;
                classNames.Add(className);
              }
            }

            return classNames;
          }
          }

            #region Windows Form Designer generated code

            /// <summary>
            /// Required method for Designer support - do not modify
            /// the contents of this method with the code editor.
            /// </summary>
            private void InitializeComponent()
            {
              this.richTextBox = new System.Windows.Forms.RichTextBox();
              this.SuspendLayout();
              //
              // richTextBox
              //
              this.richTextBox.Dock = System.Windows.Forms.DockStyle.Fill;
              this.richTextBox.Location = new System.Drawing.Point(0, 0);
              this.richTextBox.Name = "richTextBox";
              this.richTextBox.ReadOnly = true;
```

```
            this.richTextBox.Size = new
System.Drawing.Size(734, 561);
            this.richTextBox.TabIndex = 0;
            this.richTextBox.Text = "";
            //
            // FormDoc
            //
            this.AutoScaleDimensions = new
System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode =
System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new
System.Drawing.Size(734, 561);
            this.Controls.Add(this.richTextBox);
            this.FormBorderStyle =
System.Windows.Forms.FormBorderStyle.FixedS
ingle;
            this.MaximizeBox = false;
            this.MinimizeBox = false;
        namespace Graph
        {
          partial class FormMain
          {
            private
System.ComponentModel.IContainer components
= null;

          protected override void Dispose(bool
disposing)
          {
            if (disposing && (components !=
null))
            {
              components.Dispose();
            }
            base.Dispose(disposing);
          }
            this.Name = "FormDoc";
            this.ShowInTaskbar = false;
            this.StartPosition =
System.Windows.Forms.FormStartPosition.Center
Parent;
            this.Text = "Опис";
            this.Load += new
System.EventHandler(this.FormDoc_Load);
            this.ResumeLayout(false);

          }

          #endregion

          private
System.Windows.Forms.RichTextBox
richTextBox;
        }
      }
          #region Windows Form Designer
generated code
          private void InitializeComponent()
          {
            this.components = new
System.ComponentModel.Container();
            System.Windows.Forms.GroupBox
groupBoxA;

System.Windows.Forms.DataGridViewCellStyle
dataGridViewCellStyle1 = new
System.Windows.Forms.DataGridViewCellStyle()
;
            System.Windows.Forms.GroupBox
groupBoxB;

System.ComponentModel.ComponentResourceM
anager resources = new
```

```
System.ComponentModel.ComponentResourceManager(typeof(FormMain));
            System.Windows.Forms.GroupBox groupBoxResult;
            this.dataGridView = new System.Windows.Forms.DataGridView();
            this.pictureBox = new System.Windows.Forms.PictureBox();
            this.listBoxResult = new System.Windows.Forms.ListBox();
            this.buttonProcess = new System.Windows.Forms.Button();
            this.trackBarSize = new System.Windows.Forms.TrackBar();
            this.buttonSave = new System.Windows.Forms.Button();
            this.buttonLoad = new System.Windows.Forms.Button();
            this.buttonRandom = new System.Windows.Forms.Button();
            this.buttonDraw = new System.Windows.Forms.Button();
            this.toolTip1 = new System.Windows.Forms.ToolTip(this.components);
            this.buttonDoc = new System.Windows.Forms.Button();
            groupBoxA = new System.Windows.Forms.GroupBox();
            groupBoxB = new System.Windows.Forms.GroupBox();
            groupBoxResult = new System.Windows.Forms.GroupBox();
            groupBoxA.SuspendLayout();

((System.ComponentModel.ISupportInitialize)(this.dataGridView)).BeginInit();
            groupBoxB.SuspendLayout();

((System.ComponentModel.ISupportInitialize)(this.pictureBox)).BeginInit();
            groupBoxResult.SuspendLayout();

((System.ComponentModel.ISupportInitialize)(this.trackBarSize)).BeginInit();
            this.SuspendLayout();
            //
            // groupBoxA
            //

groupBoxA.Controls.Add(this.dataGridView);
            groupBoxA.Location = new System.Drawing.Point(12, 12);
            groupBoxA.Name = "groupBoxA";
            groupBoxA.Size = new System.Drawing.Size(390, 403);
            groupBoxA.TabIndex = 0;
            groupBoxA.TabStop = false;
            groupBoxA.Text = "Матриця графу";
            //
            // dataGridView
            //

this.dataGridView.AllowUserToAddRows = false;

this.dataGridView.AllowUserToDeleteRows = false;

this.dataGridView.AllowUserToOrderColumns = true;

this.dataGridView.AllowUserToResizeColumns = false;

this.dataGridView.AllowUserToResizeRows = false;
```

```
            this.dataGridView.Anchor =
((System.Windows.Forms.AnchorStyles)((((Syste
m.Windows.Forms.AnchorStyles.Top |
System.Windows.Forms.AnchorStyles.Bottom)
            |
System.Windows.Forms.AnchorStyles.Left)
            |
System.Windows.Forms.AnchorStyles.Right)));

this.dataGridView.AutoSizeColumnsMode =
System.Windows.Forms.DataGridViewAutoSizeC
olumnsMode.AllCells;

this.dataGridView.ColumnHeadersVisible = false;
            dataGridViewCellStyle1.Alignment =
System.Windows.Forms.DataGridViewContentAl
ignment.MiddleLeft;
            dataGridViewCellStyle1.BackColor =
System.Drawing.SystemColors.Window;
            dataGridViewCellStyle1.Font = new
System.Drawing.Font("Microsoft Sans Serif",
8.25F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point,
((byte)(204)));
            dataGridViewCellStyle1.ForeColor =
System.Drawing.SystemColors.ControlText;
            dataGridViewCellStyle1.Format =
"0.#";

dataGridViewCellStyle1.SelectionBackColor =
System.Drawing.SystemColors.Highlight;

dataGridViewCellStyle1.SelectionForeColor =
System.Drawing.SystemColors.HighlightText;
            dataGridViewCellStyle1.WrapMode =
System.Windows.Forms.DataGridViewTriState.F
alse;
            this.dataGridView.DefaultCellStyle =
dataGridViewCellStyle1;
            this.dataGridView.Location = new
System.Drawing.Point(6, 19);
            this.dataGridView.MultiSelect = false;
            this.dataGridView.Name =
"dataGridView";
            this.dataGridView.ReadOnly = true;
            this.dataGridView.RowHeadersVisible
= false;
            this.dataGridView.SelectionMode =
System.Windows.Forms.DataGridViewSelection
Mode.CellSelect;
            this.dataGridView.Size = new
System.Drawing.Size(378, 378);
            this.dataGridView.TabIndex = 0;

this.toolTip1.SetToolTip(this.dataGridView,
"Матрия графу, що показує \r\n зв\'язок між
вершинами \r\n");
            this.dataGridView.CellValueChanged
+=                                  new
System.Windows.Forms.DataGridViewCellEvent
Handler(this.dataGridView_CellValueChanged);
            //
            // groupBoxB
            //

groupBoxB.Controls.Add(this.pictureBox);
            groupBoxB.Location = new
System.Drawing.Point(459, 12);
            groupBoxB.Name = "groupBoxB";
            groupBoxB.Size = new
System.Drawing.Size(390, 403);
            groupBoxB.TabIndex = 1;
            groupBoxB.TabStop = false;
            groupBoxB.Text = "Схема графу";
            //
```

```
            // pictureBox
            //
            this.pictureBox.Anchor =
((System.Windows.Forms.AnchorStyles)((((Syste
m.Windows.Forms.AnchorStyles.Top |
System.Windows.Forms.AnchorStyles.Bottom)
            |
System.Windows.Forms.AnchorStyles.Left)
            |
System.Windows.Forms.AnchorStyles.Right)));
            this.pictureBox.BackColor =
System.Drawing.SystemColors.ButtonFace;
            this.pictureBox.BackgroundImage =
((System.Drawing.Image)(resources.GetObject("pi
ctureBox.BackgroundImage")));
            this.pictureBox.ErrorImage = null;
            this.pictureBox.InitialImage = null;
            this.pictureBox.Location = new
System.Drawing.Point(6, 19);
            this.pictureBox.Name = "pictureBox";
            this.pictureBox.Size = new
System.Drawing.Size(378, 378);
            this.pictureBox.TabIndex = 0;
            this.pictureBox.TabStop = false;

this.toolTip1.SetToolTip(this.pictureBox, "Схема
графу, що задана\r\nматрицею зліва\r\n");
            //
            // groupBoxResult
            //
            groupBoxResult.Anchor =
((System.Windows.Forms.AnchorStyles)((System.
Windows.Forms.AnchorStyles.Bottom |
System.Windows.Forms.AnchorStyles.Left)));

groupBoxResult.Controls.Add(this.listBoxResult);
            groupBoxResult.Location = new
System.Drawing.Point(12, 472);
            groupBoxResult.Name =
"groupBoxResult";
            groupBoxResult.Size = new
System.Drawing.Size(623, 212);
            groupBoxResult.TabIndex = 3;
            groupBoxResult.TabStop = false;
            groupBoxResult.Text = "Результати";
            //
            // listBoxResult
            //
            this.listBoxResult.FormattingEnabled
= true;
            this.listBoxResult.Location = new
System.Drawing.Point(6, 19);
            this.listBoxResult.Name =
"listBoxResult";
            this.listBoxResult.Size = new
System.Drawing.Size(611, 238);
            this.listBoxResult.TabIndex = 0;

this.toolTip1.SetToolTip(this.listBoxResult,
"Висновки");
            //
            // buttonProcess
            //
            this.buttonProcess.Anchor =
((System.Windows.Forms.AnchorStyles)((System.
Windows.Forms.AnchorStyles.Bottom |
System.Windows.Forms.AnchorStyles.Right)));
            this.buttonProcess.Enabled = false;
            this.buttonProcess.Location = new
System.Drawing.Point(641, 516);
            this.buttonProcess.Name =
"buttonProcess";
            this.buttonProcess.Size = new
System.Drawing.Size(208, 23);
            this.buttonProcess.TabIndex = 6;
```

```
            this.buttonProcess.Text =
"Проходження у ширину";

this.buttonProcess.UseVisualStyleBackColor =
true;
            this.buttonProcess.Click +=  new
System.EventHandler(this.buttonProcess_Click);
            //
            // trackBarSize
            //
            this.trackBarSize.LargeChange = 1;
            this.trackBarSize.Location =  new
System.Drawing.Point(12, 421);
            this.trackBarSize.Maximum = 17;
            this.trackBarSize.Minimum = 1;
            this.trackBarSize.Name =
"trackBarSize";
            this.trackBarSize.Size =  new
System.Drawing.Size(837, 45);
            this.trackBarSize.TabIndex = 2;
            this.trackBarSize.TickStyle =
System.Windows.Forms.TickStyle.Both;

this.toolTip1.SetToolTip(this.trackBarSize,
"Зміщення трек бару \r\nрегулює розміри
матриці");
            this.trackBarSize.Value = 1;
            this.trackBarSize.ValueChanged +=
new
System.EventHandler(this.trackBarSize_ValueCha
nged);
            //
            // buttonSave
            //
            this.buttonSave.Anchor =
((System.Windows.Forms.AnchorStyles)((System.
Windows.Forms.AnchorStyles.Bottom |
System.Windows.Forms.AnchorStyles.Right)));
            this.buttonSave.Enabled = false;
            this.buttonSave.Location =  new
System.Drawing.Point(641, 603);
            this.buttonSave.Name = "buttonSave";
            this.buttonSave.Size =  new
System.Drawing.Size(208, 23);
            this.buttonSave.TabIndex = 7;
            this.buttonSave.Text = "Записати до
файлу";

this.buttonSave.UseVisualStyleBackColor = true;
            this.buttonSave.Click +=  new
System.EventHandler(this.buttonSave_Click);
            //
            // buttonLoad
            //
            this.buttonLoad.Anchor =
((System.Windows.Forms.AnchorStyles)((System.
Windows.Forms.AnchorStyles.Bottom |
System.Windows.Forms.AnchorStyles.Right)));
            this.buttonLoad.Location =  new
System.Drawing.Point(641, 632);
            this.buttonLoad.Name =
"buttonLoad";
            this.buttonLoad.Size =  new
System.Drawing.Size(208, 23);
            this.buttonLoad.TabIndex = 8;
            this.buttonLoad.Text = "Зчитати з
файлу";

this.buttonLoad.UseVisualStyleBackColor = true;
            this.buttonLoad.Click +=  new
System.EventHandler(this.buttonLoad_Click);
            //
            // buttonRandom
            //
            this.buttonRandom.Anchor =
((System.Windows.Forms.AnchorStyles)((System.
```

```
Windows.Forms.AnchorStyles.Bottom | System.Windows.Forms.AnchorStyles.Right)));
            this.buttonRandom.Location = new System.Drawing.Point(641, 545);
            this.buttonRandom.Name = "buttonRandom";
            this.buttonRandom.Size = new System.Drawing.Size(208, 23);
            this.buttonRandom.TabIndex = 4;
            this.buttonRandom.Text = "Заповнити випадковими";

this.buttonRandom.UseVisualStyleBackColor = true;
            this.buttonRandom.Click += new System.EventHandler(this.buttonRandom_Click);
            //
            // buttonDraw
            //
            this.buttonDraw.Anchor = ((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Bottom | System.Windows.Forms.AnchorStyles.Right)));
            this.buttonDraw.Enabled = false;
            this.buttonDraw.Location = new System.Drawing.Point(641, 574);
            this.buttonDraw.Name = "buttonDraw";
            this.buttonDraw.Size = new System.Drawing.Size(208, 23);
            this.buttonDraw.TabIndex = 5;
            this.buttonDraw.Text = "Відмалювати граф";

this.buttonDraw.UseVisualStyleBackColor = true;
            this.buttonDraw.Click += new System.EventHandler(this.buttonDraw_Click);
            //
            // buttonDoc
            //
            this.buttonDoc.Anchor = ((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Bottom | System.Windows.Forms.AnchorStyles.Right)));
            this.buttonDoc.Location = new System.Drawing.Point(641, 661);
            this.buttonDoc.Name = "buttonDoc";
            this.buttonDoc.Size = new System.Drawing.Size(208, 23);
            this.buttonDoc.TabIndex = 9;
            this.buttonDoc.Text = "Опис";

this.buttonDoc.UseVisualStyleBackColor = true;
            this.buttonDoc.Click += new System.EventHandler(this.buttonDoc_Click);
            //
            // FormMain
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(861, 696);
            this.Controls.Add(this.buttonDoc);
            this.Controls.Add(this.buttonDraw);
            this.Controls.Add(this.buttonProcess);
            this.Controls.Add(this.buttonRandom);
            this.Controls.Add(this.buttonLoad);
            this.Controls.Add(this.buttonSave);
            this.Controls.Add(groupBoxResult);
            this.Controls.Add(this.trackBarSize);
            this.Controls.Add(groupBoxB);
            this.Controls.Add(groupBoxA);
```

```
            this.FormBorderStyle =
System.Windows.Forms.FormBorderStyle.FixedS
ingle;
            this.MaximizeBox = false;
            this.Name = "FormMain";
            this.StartPosition =
System.Windows.Forms.FormStartPosition.Center
Screen;
            this.Text = "Курсова робота";
            groupBoxA.ResumeLayout(false);

((System.ComponentModel.ISupportInitialize)(thi
s.dataGridView)).EndInit();
            groupBoxB.ResumeLayout(false);

((System.ComponentModel.ISupportInitialize)(thi
s.pictureBox)).EndInit();

groupBoxResult.ResumeLayout(false);

((System.ComponentModel.ISupportInitialize)(thi
s.trackBarSize)).EndInit();
            this.ResumeLayout(false);
            this.PerformLayout();

        }
```

```
        #endregion
        private    System.Windows.Forms.Button
buttonProcess;
        private
System.Windows.Forms.DataGridView
dataGridView;
        private
System.Windows.Forms.TrackBar trackBarSize;
        private  System.Windows.Forms.ListBox
listBoxResult;
        private    System.Windows.Forms.Button
buttonSave;
        private    System.Windows.Forms.Button
buttonLoad;
        private    System.Windows.Forms.Button
buttonRandom;
        private
System.Windows.Forms.PictureBox pictureBox;
        private    System.Windows.Forms.Button
buttonDraw;
        private  System.Windows.Forms.ToolTip
toolTip1;
        private    System.Windows.Forms.Button
buttonDoc;
    }
}
```

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
УКРАЇНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ НАУКИ І ТЕХНОЛОГІЙ

ВСЕУКРАЇНСЬКА НАУКОВО-ТЕХНІЧНА КОНФЕРЕНЦІЯ СТУДЕНТІВ І МОЛОДИХ УЧЕНИХ “НАУКА І СТАЛИЙ РОЗВИТОК ТРАНСПОРТУ 2023”

Дніпро
2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
УКРАЇНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ НАУКИ І ТЕХНОЛОГІЙ

ВСЕУКРАЇНСЬКА НАУКОВО-ТЕХНІЧНА КОНФЕРЕНЦІЯ СТУДЕНТІВ І МОЛОДИХ УЧЕНИХ “НАУКА І СТАЛИЙ РОЗВИТОК ТРАНСПОРТУ 2023”

27 жовтня 2023 року

ЗБІРНИК ТЕЗ

Том ІІ

Дніпро
2023

22.Сухомлин О.О. (аспірант) «Можливість використання інтелектуальних методів на окремих етапах проектування КСЗІ» (кер. доц. Остапець Д.О.)…….. 22
23.Журавльов Д.Д. (КС2011) «Можливості використання обладнання CISCO VLAN в приміщеннях кафедри ЕОМ» (кер. проф. Жуковицький І.В.)…… 23
24.Ванін М.В. (КС2221) «Використання ПЛІС в багатопроцесорних системах» (кер. доц. Шаповалов В.О.)……... 23
25.Ульянченко Д. С. (КС2011) «Проектування спрощеного двоядерного обчислювача з використанням мови VHDL» (кер. доц. Шаповалов В. О…… 24
26.Горбатов В.С. (аспірант) «Існуючі методи оптимізації пошуку атак у системах виявлення мережевих вторгнень» (кер. доц. Журба А.О.)……. 24
Підсекція «Інформаційні технології та системи» …… 25
1.Середа О.А. (ПЗ2222) «Розпізнавання рукописних символів за допомогою нейронних мереж» (кер. доц. Горячкін В.М.)……. 25

2.Сирота О. А. (аспірант) «Застосування рефакторингу для покращення продуктивності мобільних додатків» (кер. доц. Горячкін В.М.)……... 26

3.Бажин В. А. (ПЗ2222) «Дослідження патернів прив'язки даних за зчепленням» (кер. доц. Куроп'ятник О. С.)…… 27

4.Юхно Н. А. (ПЗ2221) «Перспективи використання та етичні виклики технології розпізнавання обличчя» (кер. доц. Горячкін В.М.).....28

5.Бондар Ю.О. (ДНУ ім.О.Гончара) «Перспективні напрямки удосконалення автоматизованих систем вивчення німецької мови» (кер. ас. Пузирей Н.В.) 29

6.Мурашов О.В. (аспірант) «Інформаційні технології моніторингу та реабілітації при нерівномірних інтервалах спостережень» (кер. проф. Скалозуб В.В.)……. 30

7.Терлецький І.А. (аспірант) «Інтелектуальна підтримка управління технологічними процесами залізничного транспорту за допомогою класифікації неточних даних» (кер. проф. Скалозуб В.В.)……. 31

8.Саєнко А.А. (ПЗ2222) «Покращення цілісності даних в галузі комп'ютерної інженерії та кібербезпеки: переваги технології блокчейн» (кер. доц. Гришечкіна Т.С.) …… 32
Підсекція «Автоматизація та комп'ютерно-інтегровані технології»…….. 33
1.Босий О.С. (АВ01-18м) «Система автоматичного управління тепловим режимом кільцевої печі ПАТ «Інтерпайп НТЗ» при збагаченні дуття технологічним киснем» (кер. доц. Михайловський М.В.)…… 33
2.Бутенко М.В. (АВ01-18м) «Дослідження та оптимізація системи управління транспортуванням стрижнів дрібносортного прокату секційним рольгангом» (кер. проф. Потап О.Ю.) 33
3.Долгополов Д.В. (АВ01-18м) «Розробка лабораторного комплексу з дослідження системи регулювання температури із застосуванням ПЛК «Delta» (кер. доц. Зінченко М.Д.)……... 34
4.Кравченко М.Є. (АВ01-18м) «Розробка АСУ ділянкою неруйнівного контролю якості труб» (кер. доц. Шибакінський В.І.)……. 35
5.Куделя Е.В. (АВ01-18м) «Система автоматичного регулювання рівня розплаву в ковшовому вакууматорі ПАТ «Інтерпайп НТЗ» (кер. доц. Тарасевіч І.Г.) 35
6.Соловйова К.О. (АВ01-18м) «Розробка системи дистанційного керування кроковим двигуном навчальної транспортної лінії» (кер. доц. Рибальченко М.О.) 36

- збільшення команди розробників не збільшує швидкість розробки;
- продуктивність додатку падає або від початку знаходиться на низькому рівні.

Прикладом поганої продуктивності додатку може бути повільна робота інтерфейсу користувача - низька кількість кадрів, відмальованих операційною системою за секунду часу на екрані девайсу. Окрім цього це може виражатися в смиканні анімацій, підвисання інтерфейсу, повільного завантаження, тощо.

Можливим інструментом вирішення усіх описаних проблем, окрім проблеми продуктивності, є рефакторинг. Рефакторинг коду - це процес покращення якості коду без зміни його функціональності. Процес рефакторингу може включати такі етапи, як усунення дублювання коду, видалення зайвих фрагментів коду, поліпшення читабельності та структури коду. В основному його використовують саме з метою покращення структури і полегшення розуміння програмного коду. Набагато рідше його розглядають як можливість покращення продуктивності застосунку.

Використання рефакторингу коду з метою покращення продуктивності та користувацького досвіду є важливим питанням для дослідження. Знаючи чи впливає рефакторинг на продуктивність роботи застосунку, як саме та яких проблем в цьому контексті можливо уникнути, можна більш чітко розуміти чи впливає удосконалення коду застосунку на якість його роботи. Ще одним важливим пунктом є аналіз які типи рефакторингу є більш або менш ефективним в контексті покращення користувацького досвіду, які проблеми можуть виникнути.

Результати проведеного дослідження можуть бути використані для аналізу доцільності використання такого методу покращення продуктивності програмних засобів, а також як доповнення вже існуючих даних щодо використання рефакторингу. Окрім того, це може допомогти оцінити проблеми, а також можливі ризики, які виникають під час рефакторингу та як вони мінімізуються та вирішуються. Відповіді на питання поставлені вище будуть корисним доповненням до знань не тільки досвідчених розробників мобільних додатків, а й початківцям, студентам та викладачам навчальних закладів.

ДОСЛІДЖЕННЯ ПАТЕРНІВ ПРИВ'ЯЗКИ ДАНИХ ЗА ЗЧЕПЛЕННЯМ

Бажин В. А., керівник доц. Куроп'ятник О. С.
Український державний університет науки і технологій

У сучасному програмуванні досить часто програмісти-початківці, а інколи й досвідчені спеціалісти стикаються з задачею вибору патернів під час написання коду. Виникають питання: як їх порівняти за ефективністю, чи буде той, чи інший патерн кращим у даному випадку тощо. Неможливо на практиці, без широкого досвіду застосування патернів, обрати найкращий. Тому залишається лише покладатися на власний досвід, або читати відповідну літературу, переписуючи код й перебираючи різні варіанти у пошуках найкращого. Задача вибору ефективного патерну також вирішується при розборі та використанні патернів прив'язки даних, що є досить розповсюдженими у написанні коду та являються актуальними при використанні у програмах для бізнес проєктів.

Для вирішення задачі визначення ефективності патернів прив'язки даних за зчепленням у роботі пропонується провести дослідження, яке передбачає написання відповідних тестових програм для порівняння окремих видів патернів на конкретних прикладах, що дозволить прозоріше оцінити їхню ефективність у тому чи іншому випадку.

Зчеплення було обрано за головний показник ефективності, оскільки саме він якісно показує залежність коду від архітектури (зв'язки між класами). Для вирішення

даної задачі пропонується метод на основі теорії графів, оскільки графи можна тісно пов’язати з архітектурою. Наприклад, вершини графу – це класи, а ребра – це зв’язки між ними. Застосування такого методу на основі теорії графів для написаних декількох тестових програм дозволить визначити ефективність патернів прив’язки даних за зчепленням, а також виконати детальний аналіз проблеми вибору архітектурних рішень на основі патернів, що у майбутньому може буде використано для розвинення теми критеріїв ефективності, а також застосуватися на практиці під час написання коду з використанням патернів прив’язки даних. В подальшому результати дослідження можуть бути впроваджені у навчальному процесі при вивчені дисциплін, що передбачають проєктування програмних систем, за спеціальності 121 «Інженерія програмного забезпечення».