Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Соколова Артура Вікторовича*
(ПІБ)

академічної групи *122-19ск-2*
(шифр)

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

освітньої програми *Комп'ютерні науки*
(назва освітньої програми)

на тему: *Розробка графічного редактору «UML-Diagram drawer» для створення та редагування UML-діаграм з використанням .NET*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*доц. Кабак Л.В.*			
розділів:				
спеціальний	*доц. Кабак Л.В.*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент	*доц. Шедловський І.А.*			
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2022

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

_____ І.М. Удовик
(підпис) (прізвище, ініціали)

« » _____ 2022 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *122-19ск-2* *Соколова Артура Вікторовича*
(група) (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка графічного редактору «UML-Diagram drawer» для створення та редагування UML-діаграм з використанням .NET*

затверджена наказом ректора НТУ «ДП» від «18» травня 2022 р. № 268-с

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів практик та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі.*** ***Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми.***	***13.05.2022 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки.***	***27.05.2022 р.***

Завдання видав _____ *доц. Кабак Л.В.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Соколов А.В.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2022 р.*

Термін подання кваліфікаційної роботи до ЕК: *13.06.2022 р.*

РЕФЕРАТ

Пояснювальна записка: 71 с., 19 рис., 4 табл., 3 дод., 20 джерел.

Об'єкт розробки: програмний додаток графічного редактору для створення та редагування UML-діаграм.

Мета кваліфікаційної роботи: надати користувачу сучасний ефективний інструмент для створення та редагування UML-діаграм за рахунок створення відповідного програмного продукту.

У вступі розглядається аналіз та сучасний стан проблеми, конкретизується мета кваліфікаційної роботи та галузь її застосування, наведено обґрунтування актуальності теми та уточнюється постановка завдання.

У першому розділі проведено аналіз предметної області, визначено актуальність завдання та призначення розробки, розроблена постановка завдання, задані вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі виконано аналіз існуючих рішень, обрано вибір платформи для розробки, виконано проектування і розробка програми, наведено опис алгоритму і структури функціонування підсистеми, визначені вхідні і вихідні дані, наведені характеристики складу параметрів технічних засобів, описаний виклик та завантаження застосунку, описана робота програми.

В економічному розділі визначено трудомісткість розробленої програми, проведений розрахунок вартості роботи по створенню застосунку та розраховано час на його створення.

Практичне значення полягає у створенні графічного редактору, що забезпечує користувачу ефективну роботу з UML-діаграмами.

Актуальність розроблено редактору UML-діаграм визначається великим попитом на подібні розробки.

Список ключових слів: UML-ДІАГРАМА, ГРАФІЧНИЙ РЕДАКТОР, СИСТЕМА.

ABSTRACT

Explanatory note: 71 pp., 19 fig., 4 tabs., 3 apps, 20 sources.

Development object: graphic editor software application for creating and editing UML diagrams.

The purpose of the qualification work: to provide the user with a modern effective tool for creating and editing UML-diagrams by creating the appropriate software product.

The introduction considers the analysis and current state of the problem, specifies the purpose of the qualification work and the field of its application, provides a justification for the relevance of the topic and clarifies the problem.

In the first section the analysis of the subject area is carried out, the urgency of the task and purpose of development are defined, the statement of the task is developed, requirements to software realization, technologies and software are set.

The second section analyzes the existing solutions, selects the platform for development, performs design and development of the program, describes the algorithm and structure of the subsystem, determines the input and output data, provides characteristics of the parameters of technical means, describes the call and application load, describes the program.

The economic section determines the complexity of the developed program, calculates the cost of work to create an application and calculates the time for its creation.

The practical value is to create a graphical editor that allows the user to work effectively with UML-diagrams.

The relevance of the UML chart editor developed is determined by the high demand for such developments.

Keywords: UML-DIAGRAM, GRAPHIC EDITOR, SYSTEM.

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ

ЕОМ - електронно-обчислювальна машина;

ІС - інформаційна система;

ПЗ - програмне забезпечення;

ІТ - інформаційні технології;

UML - Unified Modeling Language.

ВСТУП

UML (Unified Modeling Language)-діаграма – це графічний опис для об’єктного моделювання в області розробки програмного продукту, для моделювання бізнес процесів, системного проектування і відображення організаційних структур. У UML-діаграмах відображено нові риси бачення програми.

Існує велика кількість UML-діаграм. Розрізняють наступні види UML-діаграм:

- діаграма активностей (Activity diagram);
- діаграма класів (Class diagram);
- діаграма розгортання (Deployment diagram);
- діаграма співробітництва (Collaboration diagram);
- діаграма варіативності (UseCase diagram)[1].

На сьогодні UML-діаграми служать засобом комунікації всередині команди та при спілкуванні розробника програмного забезпечення з замовником. UML-діаграми використовуються при моделюванні архітектури великих проектів, де можна зібрати як великі, так і дрібніші деталі і створити каркас (схему) програми. На підставі цього пізніше буде будуватись код.

Більшість додатків, які автоматизованно будують UML-діаграми, з цими ж функціями, що можна знайти безкоштовно в інтернеті мають малий спектр можливостей та не дають бажаного результату. Програми, які є платними з тими ж функціямі вже мають більше можливостей, але їх складність, кількість функцій та ціна можуть відштовхнути користувачів. Наприклад, UML-діаграми Edrawsoft має широкий спектр можливостей у плані роботи з UML-діаграмами, але не кожен користувач може навчитися їх застосовувати. Серед безкоштовних додатків можна виділити додаток UML-діаграми Draw.it , але він має вузький спектр можливостей. Шрифти, які використовує програма класичні без можливості додавання нових.

Таким чином більшість програм для створення UML-діаграм мають досить невеликі можливості, функції, які зазвічай не задовільняють потребам користувачів.

Метою даної кваліфікаційної роботи є створення програмного додатку графічного редактору для створення та редагування UML-діаграм.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальні відомості з предметної галузі

Універсальна мова моделювання (Unified Modelling Language або UML) — це мова позначень або побудови діаграм, призначена для визначення, візуалізації і документування моделей зорієнтованих на об’єкти систем програмного забезпечення. UML не є методом розробки, іншими словами, у конструкціях цієї мови не повідомляється про те, що робити першим, а що останнім, і не надається інструкцій щодо побудови вашої системи, але ця мова допомагає вам наочно переглядати компонування системи і полегшує співпрацю з іншими її розробниками. Розробкою UML керує Object Management Group (OMG). Ця мова є загальноприйнятим стандартом графічного опису програмного забезпечення.

UML розроблено для розробки структури зорієнтованого на об’єкти програмного забезпечення, ця мова має дуже обмежену користь для програмування на основі інших парадигм.

Конструкції UML створюються з багатьох модельних елементів, які позначають різні частини системи програмного забезпечення. Елементи UML використовуються для побудови діаграм, які відповідають певній частині системи або точці зору на систему. У Umbrello UML Modeller реалізовано підтримку таких типів діаграм:

- Діаграма випадків використання показує дієвих осіб (людей або інших користувачів системи), випадки використання (сценарії використання системи) та їх взаємодію.
- Діаграми класів, на яких буде показано класи та зв’язки між ними.
- Діаграми послідовності, на яких показано об’єкти і послідовність методів, якими ці об’єкти викликають інші об’єкти.
- Діаграми співпраці, на яких буде показано об’єкти та їх взаємозв’язок з наголосом на об’єкти, які беруть участь у обміні повідомленнями.

- Діаграми стану, на яких буде показано стани, зміну станів і події у об’єкті або частині системи.
- Діаграми діяльності, на яких буде показано дії та зміни однієї дії іншою, які є наслідком подій, що сталися у певній частині системи.
- Діаграми компонентів, на яких буде показано програмні компоненти високого рівня (на зразок KParts або Java Beans).
- Діаграми впровадження, на яких буде показано екземпляри компонентів та їх взаємодію.
- Діаграми взаємозв’язку сутностей, на яких буде показано дані, взаємозв’язки і умови обмеження зв’язків між даними.

1.2. Призначення розробки та область застосування

Основне призначеня графічного редактора, що розробляється в роботі, це створення та редагування UML-діаграм.

Програма повинна мати наступні функції:

– можливість створення або малювання основних об'єктів UML діаграм(класів, інтерфейсів та стрілок);

– можливість вибору кольору, товщини, шрифту тексту в класі або інтерфесі;

– можливість переміщення вже доданих об'єктів, а саме класів, інтерфейсів, стрілок;

– можливість зміни існуючих класів, інтерфейсів та стрілок (розмір, товщина, колір, шрифт, текст, тип стрілок);

– можливість видалення або редагуваня об'єктів, якщо користувач помилково додав об’єкт у UML-діаграму;

– можливість збереження, завантаження схем у програмному додатку (збереження в тому числі картинкою) форматів JPEG або UML-діаграм;

– автоматичне підлаштування об'єкту класу, а саме розмір виходячи з вмісту, а також стрілок в залежності від розташування класів та інтерфейсів;

– множинне виділення об'єктів з наступним переміщенням або видаленням;

– скасування останніх дій.

1.3. Підстава для розробки

Відповідно до освітньої програми, згідно навчального плану та графіків навчального процесу, в кінці навчання студент виконує кваліфікаційну роботу (проект). Тема роботи узгоджується з керівником проекту, випускаючою кафедрою, та затверджується наказом ректора.

Отже, підставами для розробки (виконання кваліфікаційної роботи) є:

- освітня програма спеціальності 122 «Комп’ютерні науки»;
- навчальний план та графік навчального процесу;
- наказ ректора Національного технічного університету «Дніпровська політехніка» № 268-с від 18.05.2022 р;
- завдання на кваліфікаційну роботу на тему: «Розробка графічного редактору «UML-Diagram drawer» для створення створення та редагування UML-діаграм з використанням .NET.».

1.4. Постановка завдання

Спроектувати та розробити графічний редактор для створення та редагування UML-діаграм, за допомогою мови програмування C# у середовищі програмування Microsoft VisualStudio 2019 з використанням .NET.

Результат роботи програми має зберігатися у окремому файлі, у обраному користувачем місці.

Головне вікно програми повино складатися з головного меню, панелі інструментів, робочої області та рядку стану.

Програма-редактор повинна мати наступні функції:

– можливість створення або малювання основних об'єктів UML діаграм(класів, інтерфейсів та стрілок);

– можливість вибору кольору, товщини, шрифту тексту в класі або інтерфесі;

– можливість переміщення вже доданих об'єктів, а саме класів, інтерфейсів, стрілок;

– можливість зміни існуючих класів, інтерфейсів та стрілок (розмір, товщина, колір, шрифт, текст, тип стрілок);

– можливість видалення або редагуваня об'єктів, якщо користувач помилково додав об’єкт у UML-діаграму;

– можливість збереження, завантаження схем у програмному додатку (збереження в тому числі картинкою) форматів JPEG або UML-діаграм;

– автоматичне підлаштування об'єкту класу, а саме розмір виходячи з вмісту, а також стрілок в залежності від розташування класів та інтерфейсів;

– множинне виділення об'єктів з наступним переміщенням або видаленням;

– скасування останніх дій.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Основне функціональне призначеня графічного редактора, що розробляється в роботі це створення та редагування UML-діаграм.

Програма редактор повинна мати наступні функції:

– можливість створення або малювання основних об'єктів UML діаграм(класів, інтерфейсів та стрілок);

– можливість вибору кольору, товщини, шрифту тексту в класі або інтерфесі;

– можливість переміщення вже доданих об'єктів, а саме класів, інтерфейсів, стрілок;

– можливість зміни існуючих класів, інтерфейсів та стрілок (розмір, товщина, колір, шрифт, текст, тип стрілок);

– можливість видалення або редагуваня об'єктів, якщо користувач помилково додав об’єкт у UML-діаграму;

– можливість збереження, завантаження схем у програмному додатку (збереження в тому числі картинкою) форматів JPEG або UML-діаграм;

– автоматичне підлаштування об'єкту класу, а саме розмір виходячи з вмісту, а також стрілок в залежності від розташування класів та інтерфейсів;

– множинне виділення об'єктів з наступним переміщенням або видаленням;

– скасування останніх дій.

1.5.2. Вимоги до інформаційної безпеки

Для уникнення некоректної роботи програми необхідно реалізувати:

– семантичний та синтаксичний контроль вхідних даних;

– обробку виняткових ситуацій;

– виведення повідомлень про помилки;

– можливість повторного введення даних;

– можливість безперервної роботи протягом не менше 120 годин (5 діб);

– платформну незалежність.

– час відновлення після збою - 10 хв;

– вірогідність виникнення не більше 2 логічних помилок на 1000 операторів за 1 рік експлуатації;

– забезпечення неушкодженого стану даних, що зберігаються в базі даних, у випадку відмови підсистеми.

1.5.3. Вимоги до складу та параметрів технічних засобів

Програма-редактор не має бути вимогливою до складу та параметрів технічних засобів та має працювати на більшості сучасних ЕОМ під управлінням ОС Windows чи Linux. Для роботи з UML-діаграмами та графікою в цілому необхідно мати персональний комп’ютер (далі-ПК), для швидкого відображення пікслеві на полотні та єкрані. Для цього необхідно мати відеокарту з високою частотою оновлення пам’яті та гарним об’ємом шини.

Для роботи програми необхідна опертавна пам'ять у великій кількості, через запам’ятовування програмою дій користувача, у разі помилки мати можливість повернутися на дію назад, список дій зберігається у оперативній пам’яті.

Персональний комп’ютер, на якому буде працювати програма повинен мати такі характеристики:

- ОС Windows або Linux;
- процесор два ядра по 1.5ГГц чи краще;
- оперативна пам’ять 512Мб чи більше;
- відеоадаптер з вбудованою пам’яттю 128 Мб чи більше;
- вільний простір на жорстоку диску 1 Гб.

Наведені вище технічні характеристики є рекомендованими, тобто при наявності технічних засобів не нижче зазначених, розроблений програмний виріб буде функціонувати відповідно до вимог щодо надійності, швидкості обробки даних і безпеки, висунутими замовником.

1.5.4. Вимоги до інформаційної та програмної сумісності

Розроблений аналог графічного редактора для створення UML-діаграм може повноцінно працювати в операційній системі Windows починаючи з версії XP або Linux.

В даний час Microsoft Windows є найпопулярнішою ОС, не тільки через зручність для використання і функціонал, а й через високий рівень інтеграції з можливостями її ядра і іншого програмного забезпечення, в тому числі Microsoft Office.

В якості комплексної операційної системи, всі версії Microsoft Windows поставляються з попередньо встановленим програмним забезпеченням, яке можна використовувати відразу після її установки. Основні текстові редактори і калькулятор стали доступними для використання в найпершій версії Windows. Windows 98 внесла Media Payer, Internet Explorer і Outlook Express. Починаючи з Windows Vista з'явилися фотогалерея DVD Maker і бічна панель, яка в Windows 7, 8, 10 представлена гаджетами-інформерамі. Починаючи з другого пакета поновлення Windows XP і з виходом Windows Vista з'явилася вбудована функція безпеки – фаєрвол (брандмауер). Ця функціональність у сучасних ОС Windows реалізується на рівні ядра.

Також Microsoft випустила серверні версії, мобільні і вбудовані версії своєї операційної системи. Система Windows Server має конкурентний адміністративний пакет, включає в себе послуги з оновлення, сервер зберігання, веб-сервер, медіа сервер, тощо.

З кожним днем Microsoft Windows випускає нововведення і оновлення безпеки. Багатьох користувачів цілком влаштовує можливість роботи в цьому логічно побудованому сімействі операційних систем Windows.

Розглянемо особливості Windows-подібних операційних систем:

– багатозадачність, Windows може забезпечити одночасне і незалежне один від одного виконання декількох програм;

– єдиними для Windows, організовано так, що є можливість створювати дані в одних програмах і переносити їх в інші програми;

– єдиний інтерфейс користувача, Windows забезпечує однаковий інтерфейс для всіх своїх компонентів, а також для програм, які розроблені для роботи під управлінням Windows;

– єдиний апаратно-програмний інтерфейс, ОС Windows забезпечує сумісність різноманітного обладнання та програм;

– графічний інтерфейс користувача, як елемент управління використовується маніпулятор «миша», файли, папки відображаються на екрані у вигляді графічних значків;

– стандарт самовстановаих пристроїв (Plug and Play).

Розглянемо особливості Linux -подібних операційних систем:

– безкоштовна операційна система;

– вихідні коди ядра дозволяють удосконалювати операційну систему всім бажаючим;

– усі бажаючі можуть розвивати Linux, не побоюючись судових позовів;

– може працювати, як на кишеньковому комп'ютері, так і на суперкомп'ютері і навіть у багатьох побутових приладах.

Підключення пристроїв відбувається автоматично. ОС Windows та Linux, визначає, які драйвера встановлені і налаштовані на роботу з новим обладнанням.

Інтерфейс Windows на відміну від Linux, як і сама система в цілому, побудований відповідно до принципів об'єктного підходу. Основними елементами призначеного для користувача інтерфейсу операційної системи Windows є такі об'єкти: робочий стіл, вікна, значки, панелі, меню, папки, програми та документи. До об'єктів відносяться також будь-які апаратні і програмні ресурси комп'ютера. Та й комп'ютер в цілому теж вважається об'єктом.

Для повноцінної роботи програми необхідно мати Windows, Linux - подібну операційну систему та комп’ютер з необхідною потужністю.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Функціональне призначення системи

Основне функціональне призначеня графічного редактора, що розробляється в роботі це створення та редагування UML-діаграм.

Програма редактор реалізує наступні функції:

– можливість створення або малювання основних об'єктів UML діаграм(класів, інтерфейсів та стрілок);

– можливість вибору кольору, товщини, шрифту тексту в класі або інтерфесі;

– можливість переміщення вже доданих об'єктів, а саме класів, інтерфейсів, стрілок;

– можливість зміни існуючих класів, інтерфейсів та стрілок (розмір, товщина, колір, шрифт, текст, тип стрілок);

– можливість видалення або редагуваня об'єктів, якщо користувач помилково додав об’єкт у UML-діаграму;

– можливість збереження, завантаження схем у програмному додатку (збереження в тому числі картинкою) форматів JPEG або UML-діаграм;

– автоматичне підлаштування об'єкту класу, а саме розмір виходячи з вмісту, а також стрілок в залежності від розташування класів та інтерфейсів;

– множинне виділення об'єктів з наступним переміщенням або видаленням;

– скасування останніх дій.

Головними компонентами для роботи з програмою є стандартні компоненти ОС Windows або Linux. Для роботи з програмою не потрібне встановлення спеціального програмного забезпечення. Вхідними данними редактора можуть бути тільки файли формату UML-діаграм. Для перегляду

результату роботи програми можна використовувати будь який графічний редактор, який підтримує вище згаданий формат, наприклад draw.it, тощо.

Даний програмний продукт є сумісний з багатьма подібними програмами, напріклад з:

- Diagrams.net — зручний сервіс для створення блок-схем, UML-діаграм, моделей бізнес-процесів онлайн. Сумісний з більшістю популярних інструментів, включаючи Google Docs, Git, Dropbox, OneDrive та інші. Є платним додатком для Windows та Unix- подібних систем. Має один із самих великих функціоналів та представляє себе програмою флагманом на всесвітньому ринку(рис. 2.1).

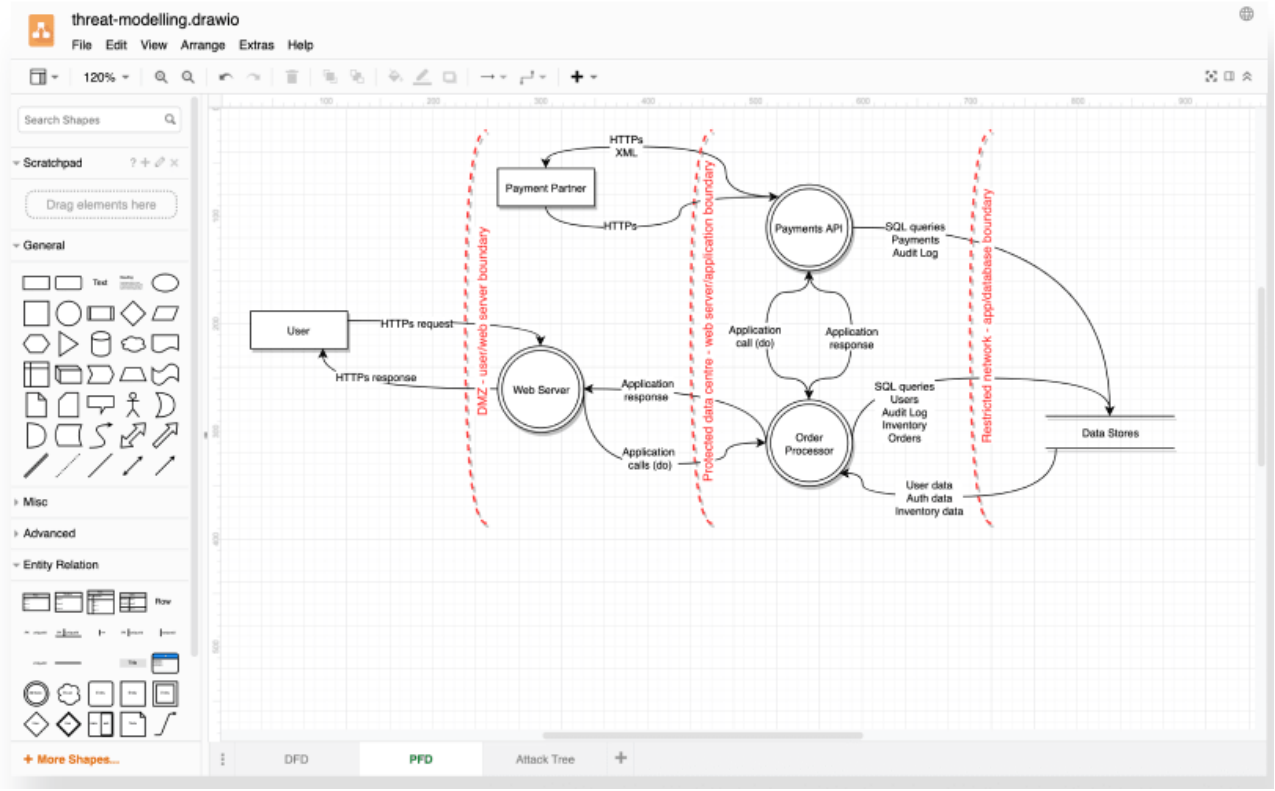

Рис. 2.1. Редактор diagrams.net

- Dbdiagram.io — додаток для побудови діаграми зв'язків для баз даних. Гарний інструмент для розробників і аналітиків (рис. 2.2).

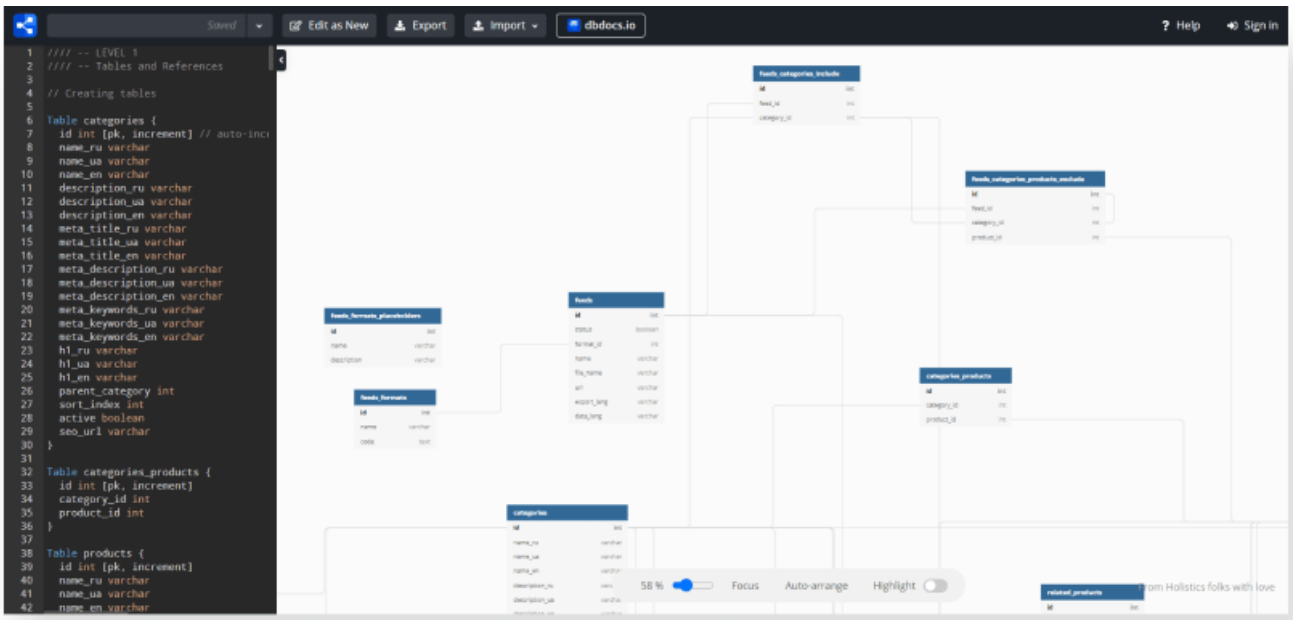

Рис. 2.2. Dbdiagram.io — додаток для побудови діаграм

- Google Drawings — безкоштовний інструмент для створення блок-схем і діаграм в складі Google Drive (менш зручний порівняно з diagrams.net). Google Drawings, частина Google Drive, дозволяє користувачам спільно створювати, обмінюватися і редагувати зображення або малюнки. Його можна використовувати для створення діаграм, дизайнів, блок-схем. Він містить підмножину функцій в Google Slides, але з різними шаблонами (рис. 2.3).

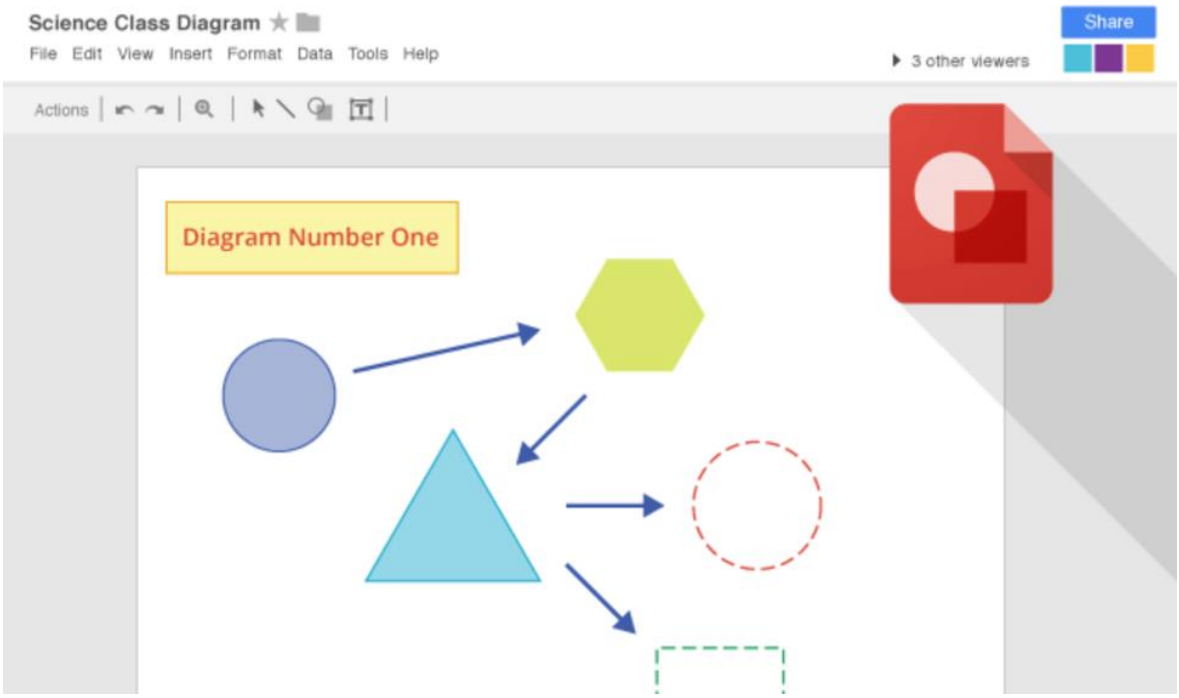

Рис. 2.3. Редактор Google Drawing

- xmind.net — програма для побудови інтелектуальних карт (mind map), логічних схем, складних структур, проведення брейнстормів і не тільки. Він надає багатий набір різних стилів візуалізації і дозволяє ділитися створеними інтелектуальними картами через їх веб-сайт. Додаток доступний у вигляді версії з відкритим вихідним кодом і платній комерційній версії «Pro». XMind Pro пропонує розширені функції, такі як діаграми Ганта, режим презентації, функції експорту (PDF, Office, MindManager і Freemind), аудіозаписи, можливість злиття, приватний онлайн-обмін і багато іншого. В іншому випадку версія з відкритим вихідним кодом є дуже функціональним і потужним програмним забезпеченням «Mind Managing». XMind автоматично вибере мову відповідно до настройками вашої ОС (рис. 2.4).

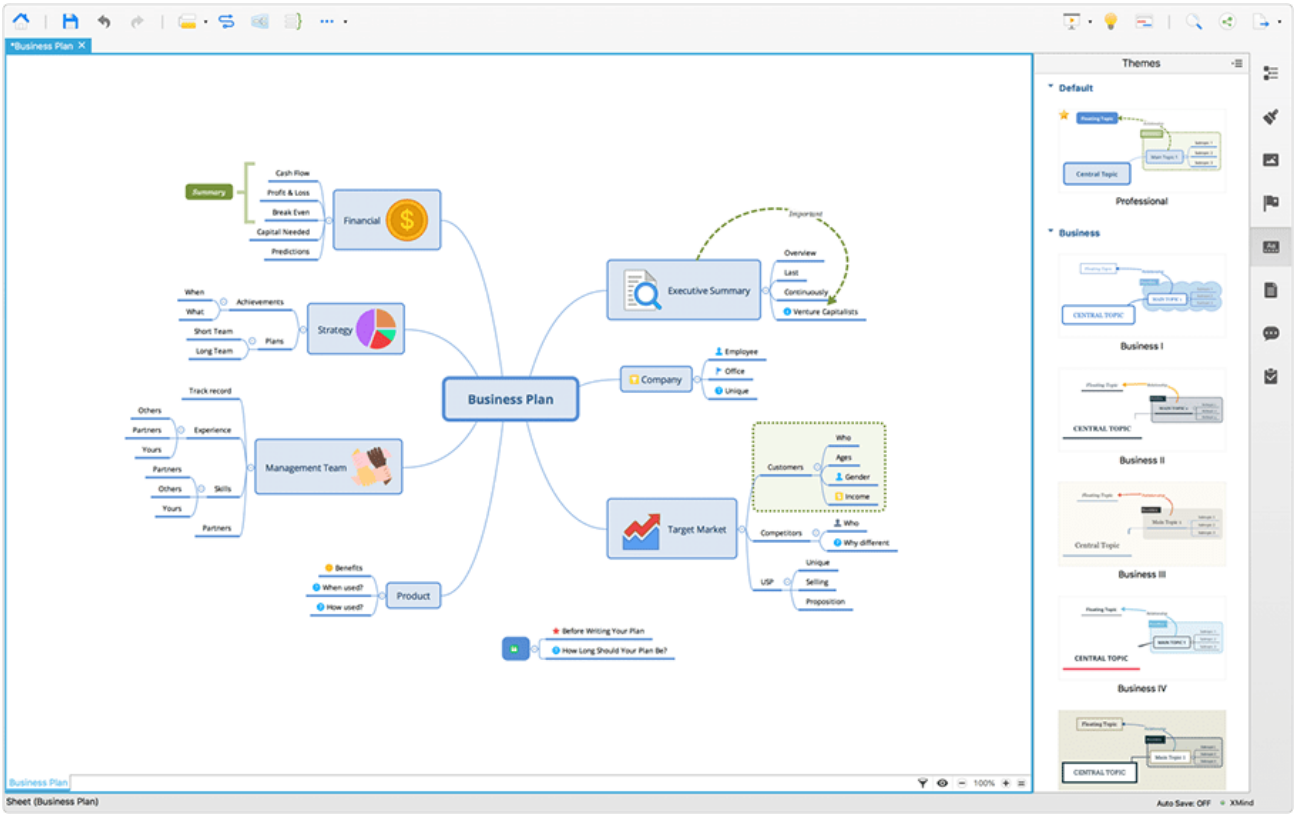

Рис. 2.4. Редактор xmind.net

2.2. Опис застосованих математичних методів

Оскільки особливості предметної області розв'язуваної задачі не передбачають застосування спкціальних математичних методів, при розробці математичні методи не використовувалися.

2.3. Опис використаних технологій та мов програмування

З появою візуального програмування створювати проекти стало набагато простіше. Виконання великих проектів скоротилося з декількох років до місяців. Візуальне програмування покращило якість програм та відкрило нові можливості у програмуванні. Середовище програмування надає форму, де можна розмістити необхідні компоненти.

Для розробки програмного додатку була обрана мова програмування C# та середовище розробки Microsoft Visual Studio 2019.

C# (вимовляється Сі-шарп) – об'єктно-орієнтована мова програмування з безпечною системою типізації для платформи.NET. Розроблена Андерсом Гейлсбергом, Скотом Вілтамутом та Пітером Гольде під егідою Microsoft Research (при фірмі Microsoft).

Мова має сувору статичну типізацію, підтримує поліморфізм, перевантаження операторів, вказівники на функції-члени класів, атрибути, події, властивості, винятки, коментарі у форматі XML. Перейнявши багато що від своїх широко відомих та використовуваних попередників – мов C++, Delphi, Модула і Smalltalk – C#, спираючись на практику їхнього використання, виключає деякі моделі, що зарекомендували себе, як проблематичні при розробці програмних систем, наприклад, множинне спадкування класів (на відміну від C++).

C# має явну підтримку ковариації та контраваріантності в родових типах, на відміну від C++, яка має певний рівень підтримки контраваріантності просто через семантику типів, що повертаються, на віртуальні методи. Члени перерахування розміщуються в своєму власному обсязі.

Мова C# не допускає глобальних змінних або функцій. Всі методи і члени повинні бути оголошені всередині класів. Статичні члени відкритих класів можуть замінювати глобальні змінні та функції.

C# створювалася, як мова компонентного програмування, і в цьому одне з головних переваг мови, спрямоване на можливість повторного використання створених компонентів. З інших об'єктивних факторів відзначимо наступні, що C# створювалася паралельно з каркасом Framework Net і повною мірою враховує всі її можливості. C# є повністю об'єктно-орієнтованою мовою, де навіть типи, вбудовані в мову, представлені класами. C# є потужною об'єктною мовою з можливостями спадкування й універсалізації. C# є спадкоємицею мов C/C++, зберігаючи кращі риси цих популярних мов програмування; завдяки каркасу Framework.Net, що стали надбудовою над операційною системою, програмісти C# одержують ті ж переваги роботи з віртуальною машиною, що й програмісти Java. Ефективність коду навіть підвищується, оскільки виконавче середовище являє собою компілятор проміжної мови, у той час, як віртуальна Java-машина є інтерпретатором байта-коду. Потужна бібліотека каркасів підтримує зручність побудови різних типів додатків на C#, дозволяючи легко будувати Web-служби, інші види компонентів, досить просто зберігати й одержувати інформацію з бази даних й інших сховищ даних, реалізація, що сполучає побудову надійного й ефективного коду, є немаловажним чинником, що сприяє успіху C#.

Подібно до Java, C# отримали наступні концепції:

– віртуальна машина – платформа .Net виконує програму подібно до віртуальної машини від Java;

– байт-код – програмний код компілюється в проміжкову мову, MSIL (Microsoft Intermediate Language), а вже потім перетворюється на машинну мову, в залежності від платформи на якій запускається програма;

– керований код – оскільки програми, написані на C#, виконуються виключно у віртуальному середовищі, це дає змогу контролювати виконання програми та у будь який момент зупинита її, а також контролювати використання

пам’яті програмою, за необхідності - збільшувати, чи видаляти частини пам’яті, які використовує програма.

Отже, мова програмування C#, у якій поєднуються потужність і гнучкість універсальних мов програмування з високою ефективністю виконавчого коду й можливістю безпосереднього доступу до апаратних ресурсів комп’ютера – одна з найкращих мов програмування.

Ключові особливості мови C#:

– компонентна орієнтованість;

– код зібраний воєдино (декларації і реалізації об'єднані разом);

– уніфікована система типів і їх безпечність;

– автоматична і мануальна робота за пам'яттю;

– використання єдиної бібліотеки класів – CLR (Common Language Runtime).

Мова C# та платформа .Net постійно розвивається, вбираючи в себе найкращі практики використання мов програмування..Net дозволяє розробляти програмне забезпечення під різні операційні системи: Windows, MacOS, Linux, Android, iOS. Mono – реалізація .Net для Linux.

Для програмування на мові C# існують середовища програмування, наприклад:

- Visual Studio;

- Visual Studio Code;

- Unity3D;

- Xamarin;

- SharpDevelop;

- Mono Develop.

Усі ці середовища покривають всі операційні системи і технології, та дозволяють розробляти мобільні, настільні(desktop), Web, вбудовані(embedded) додатки та сервіси. Бібліотеки класів .Net Framework включають в себе основу для багатьох технологій, які розробники можуть використовувати при створенні продуктів. Зокрема Entity Framework, ADO.Net – технології, що

використовуються для роботи з даними. Технології WindowsForms, WPF, ASP.Net, ASP.Net MVC – для створення desktop та Web додатків. Найбільш розповсюдженим середовищем програмування для C# є Microsoft Visual Studio.

Microsoft Visual Studio (далі-Visual Studio) – серія продуктів фірми Microsoft, які включають інтегроване середовище розробки програмного забезпечення та ряд інших інструментальних засобів. Ці продукти дозволяють розробляти, як консольні програми, так і програми з графічним інтерфейсом, в тому числі з підтримкою технології Windows Forms, а також веб-сайти, веб-застосунки, веб-служби, які розташовані в рідному середовищі, так і в керованому коді для всіх платформ, що підтримуються Microsoft Windows, Windows Mobile, Windows Phone, Windows CE, .NET Framework, .NET Compact Framework та Microsoft Silverlight. Це середовище є зручним і зрозумілим для більшості користувачів.

Воно має зручний інтерфейс, можливість зміни стилю вікна. Серед переваг можна зазначити велику кількість можливостей розробки та візуалізацію попереднього перегляду програми, що буде створена (рис. 2.5).

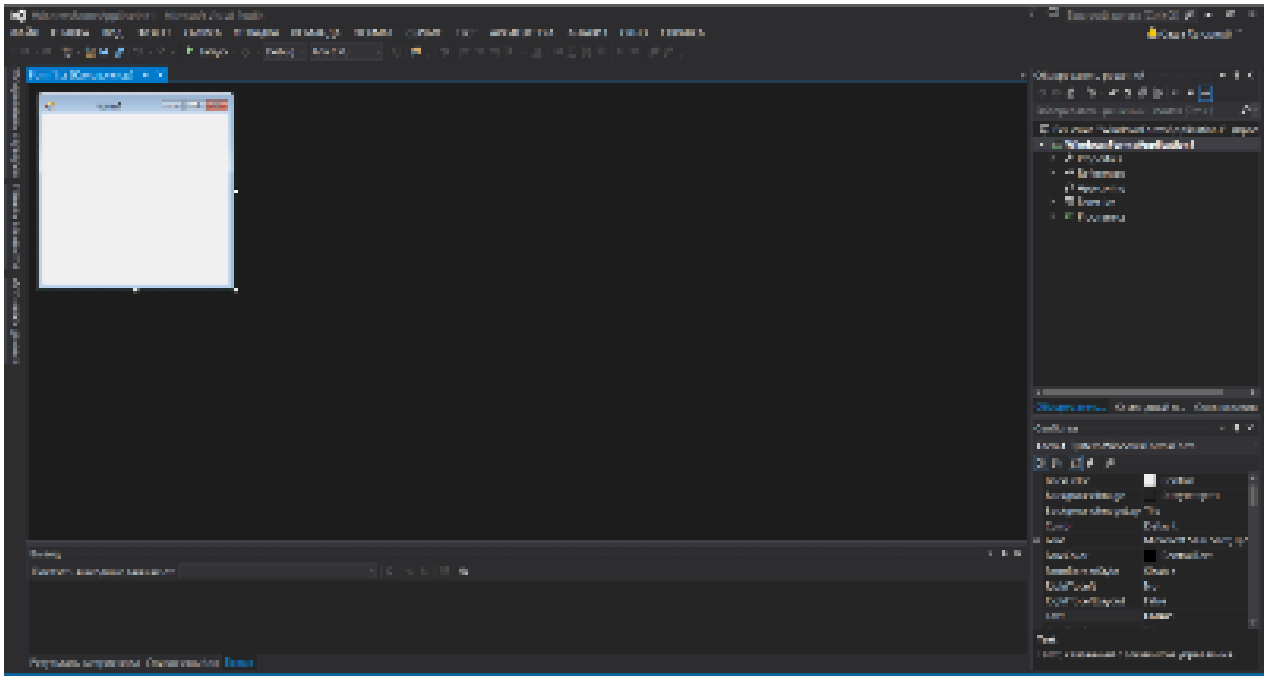

Рис. 2.5. Головне вікно середовища програмування Microsoft Visual Studio 2019

У середовищі Visual Studio 2019 приділено велику увагу правельності написання коду для зменьшення ризиків та помилок у роботі програми.

Для цього використано наступні методи. Перевірка при наборі (введенні) коду. При введенні коду в редакторі, крім перевірок його синтаксичної правильності (наприклад, вказівки на незамкнені дужку, відсутню крапку з комою і т.д.), виконуються також перевірки на розрахункові помилки, семантичну коректність коду.

Наприклад, діагностується введення недосяжного коду (unreachable code) – якщо введений оператор if, в якому частина коду в одній з альтернатив, перед якою стоїть оператор break, виявляється недосяжною. У таких випадках на вкладці повідомлень про помилки видаються відповідні попередження, помічені жовтим окличним знаком. Тобто це код, який може ніколи не використовуватися у програмі.

Якщо на мові C# в коді описана змінна x без ініціалізації, а потім вона використовується, компілятор видає повідомлення про помилку. Для аналізу коду у головному меню середовища є спеціальний пункт Analyze (розташований праворуч від пункту Architecture). Він містить ряд дій з аналізу коду, важливих як з точки зору надійних і безпечних обчислень, так і з точки зору перевірки якості коду (наприклад, його ефективності).

Для тестування і аналіз результатів тестування коду за допомогою інструментів Visual Studio використовується генератор unit-тестів NUnit (точніше, цей інструмент може бути додатково інстальований в середу Visual Studio, як зовнішній програмний пакет за допомогою утиліти NuGet) і засоби запуску unit-тестів, управління їх запуском і аналізу результатів тестування. Є і інші генератори тестів для середовища Visual Studio. В даний час тестування до сих пір залишається основним методом верифікації (перевірки правильності) коду, дуже важливою з точки зору надійних і безпечних обчислень. Слід врахувати, що в текстах довідки і підтримки (help) і в документації по Visual Studio термін верифікація (verification) означає саме тестування, а не формальну верифікацію (доказ відповідності реалізації коду його формальної специфікації).

2.4. Опис структури системи та алгоритмів її функціонування

Для розробки програми по створенню UML-діаграм C# має набір зручних бібліотек та функцій. Графічний інтерфейс додатків C#, Graphics Device Interface(GDI) - це інтерфейс Windows для представлення графічних об'єктів і передачі їх на пристрої відображення, такі, як монітори і принтери

GDI відповідає за малювання ліній і кривих, відображення шрифтів і обробку палітриґ, як і інших додатків, призначених для роботи в рамках Microsoft .NET Framework, складається з набору класів, що об'єднуються простором імен. Одне з основних просторів імен GDI мови C# є простір імен System.Drawing. Класи цього простору імен визначають перелік об'єктів і інструментів, призначених для «малювання». До найбільш часто використовуваних класів простору імен System.Drawing відносяться наступні компоненти.

Brush (Brushes, SolidBrush і т.д) – пензель використовуються для заповнення простору всередині геометричних фігур, це абстрактний базовий клас, інші типи є похідними від Brush і визначають різні набори можливостей.

Pen (Pens, SystemPens) – перо, це об'єкт класу, за допомогою якого можна малювати прямі і зигзаго подібні лінії. У класі Pen визначено набір статичних властивостей, за допомогою яких можна отримати об'єкт Pen із заданими властивостями (наприклад, з встановленим кольором).

Point (PointF) – структури, які забезпечують роботу з координатами точки. Point працює зі значеннями типу int, а PointF - зі значеннями типу float. В даній програмі Point і Pen використовується при малюванні стрілок, наприклад:

```
if ((endPoint.Location.Y < startPoint.Location.Y && endPoint.Side == Side.Up)
 || (endPoint.Location.Y > startPoint.Location.Y && endPoint.Side == Side.Bottom))
                    {
                       Points = new Point[5];
                       Points[0] = startPoint.Location;
                       Points[1] = new Point(middleX, startPoint.Location.Y);
                       y = endPoint.Location.Y < startPoint.Location.Y
                          ? endPoint.Location.Y - indentFromBorder : endPoint.Location.Y
+ indentFromBorder;
```

```
            Points[2] = new Point(middleX, y);
            Points[3] = new Point(endPoint.Location.X, y);
            Points[4] = endPoint.Location;
        }
```

В данному фрагметі коду компонент Point відповідає за малювання стрілки по чотирьом точкам, де перша і остання точки– це з‘єднання з формами, а дві інші точки – перетин повороту лінії.

Font (FontFamily) – об'єкти типу Font визначають характеристики шрифту (ім'я, розмір, накреслення, тощо), для класів, інтерфейсів. FontFamily представляє набір шрифтів, які відносяться до одного сімейства, але мають деякі невеликі відмінності.

Розглянемо приклад:

```
if (_mainData.SelectForm != null)
        {
           FontDialog.ShowDialog();
           _mainData.SelectForm.SetFont(FontDialog.Font);
           _mainData.SaveChanges();
           _mainData.PictureBoxMain.Invalidate();
        }
        else
        {
           throw new ArgumentNullException("Object is null");
        } .
```

Цей фрагмент коду демонструє використання компоненту Font у даній програмі, а саме шрифтів FontFamily, для зміни шрифту текста у формах.

У просторі імен System.Drawing також знаходяться класи Icon, Image, Color, Bitmap та інші класи так, чи інакше пов'язані з відображенням графічної інформації на екрані монітора.

Основним класом для «малювання» в мові C# є клас Graphics. Він призначений для виведення графічної інформації в клієнтську частину форми додатка. Для того щоб можна було малювати у вікні, необхідно отримати або створити для цього вікна об'єкт класу Graphics. Далі, користуючись властивостями і методами цього об'єкта, додаток може малювати у вікні різні фігури або текстові рядки, а саме класи, інтерфейси та стрілки [4].

```
public static class MainGraphics
   {
      private static Graphics _graphics;
      public static Graphics Graphics
      {
         get
         {
            if (_graphics != null)
            {
               return _graphics;
            }

            throw new ArgumentNullException("Graphics is null");
         }
         set
         {
            if(value != null)
            {
               _graphics = value;
            }
            else
            {
               throw new ArgumentNullException("Value is null");
            }
         }
      }
   }
```

В данному фрагметі коду компонент Graphics відповідає за введеня усієї графічної інформації на панель програми, а саме класів, інтерфесів і стрілок.

Робота з графічними складовими відбуватиметься за допомогою змін даних на полотні у робочій області. Це відбувається при зміні кольору певних пікселів на зображенні. Наприклад, піксель з координатами [50,48] буде білого кольору, а сусідні пікселі з ним чорного. Для утворення суцільного чорного кола, необхідно щоб пікселі в певному діапазоні були однакового кольору. Якщо колір цих пікселів не співпадає з бажаним, міняємо колір на потрібний.

Для роботи з файлами зазвичай використовують JavaScript Object Notation (JSON), у просторі імен System.Text.Json. Ключовим типом є клас JsonSerializer, який і дозволяє серіалізовать об'єкт в json і, навпаки, десеріалізовать код json в об'єкт C#.

Для збереження об'єкта в json в класі JsonSerializer визначено статичний метод aveFile (), який має ряд перевантажених версій, для обробки різних типів даних, в якому використовується функція бібліотеки json StreamWriter(), яка визиває діалогове вікно і зберігає об’єкт у задоному форматі. Аналогічним способом працює фунукція OpenFile(), яка відкриває файли формату UML-діаграм за допомогою функції new StreamReader(), напріклад:

```
using System.Text.Json;
namespace UML_Diagram_drawer
{
   public static class SaveAndLoad
   {
      private static string _splitter = "123ImSplitter!PleaseDontTouchMe!321";

      public static void SaveFile(string path, string fileDataForms, string
fileDataArrows)
      {
         using (StreamWriter saveSW = new StreamWriter(path, false))
         {

saveSW.WriteLine($"{fileDataForms}{Environment.NewLine}{_splitter}{Environ
ment.NewLine}{fileDataArrows}");
         }
      }

      public static string[] OpenFile(string path)
      {
         string[] fileData = new string[] { String.Empty };
         using (StreamReader openSR = new StreamReader(path))
         {
            fileData = openSR.ReadToEnd().Split(new[] { _splitter },
StringSplitOptions.None);
         }
         return fileData;
      }
   }
      }
```

Даний фрагмет коду відповідає за відкриття і збереження файлів за допомогою простору імен System.Text.Json, та його головних функцій OpenFile() та StreamWriter().

Для створення проекту UML_Builder були використані мова програмування C# та середовище розробки Microsoft VisualтStudio 2019.

Для початку необхідно створити проект, форми та дерево файлів проекту. C# використовує визначену та зручну систему розподілу файлів проекту. Вона представлена у вигляді дерева каталогу з можливістю додавання чи змін окремих гілок чи дерева в цілому. У стандартному вигляді дерево має наступні гілки “Рішення”, яке згодом модифікується в залежності від побудови програми.

У папці проекту UML_Builder є певний набір ключових елементів:

− файл Arrows.cs – містить відомості про усі види позначення - стрілка;

− файл Factory.cs – містить інструкцію про Arrow.cs і Form;

− папка form – містить відомості про усі види класів/інтерфейсів;

− файл AbstractModules.cs – містить абстрактний клас, в якому зберігаються усі данні про поля в формі, його розмір, стіль, тощо;

– FieldModuls.cs – описує сутності конкретного модуля полів форми;

– MetodModule.cs – описує сутності конкретного модуля методів форми;

– TitleModuleType.cs – описує сутності конкретного модуля заголовку форми;

– папка Handlers – папка зберігає інструкцію по обробці дій на формі;

– файл ArrowEditorHandler.cs – містить інструкцію дій користувача для редагування стрілок;

– файл FormEditorHandler.cs – зберігає інструкції подій для редагування форм;

– файл iEditorHandler.cs – зберігає інтерфейс хендлера редактування об’єктів;

– файл DrawArrowMouseHandler.cs – описує дію для малювання стрілок;

– DrawFromMouseHandler.cs – описує дію для малювання форм;

– iMouseHandler.cs – зберігає інтерфейс хендлера для малювання об’єктів;

– MoveAndSelectMoseHandler.cs – описує дію виділення і зсуву об’єктів;

– Default.cs – містить набір дефолтних параметрів;

– Form1.cs – базовий клас, який описує работу базової форми;
– TextField.cs – описує сутності текстового поля у формі;
– FormType.cs – зберігає лічильник типів форм;
– Form.cs – описує сутності;
– ContactPoint.cs – зберігає координати точок поєднання;
– AbstractForm.cs- зберігає усі дані для форми.

Перелік усіх файлів проекту наведено на рис. 2.6.

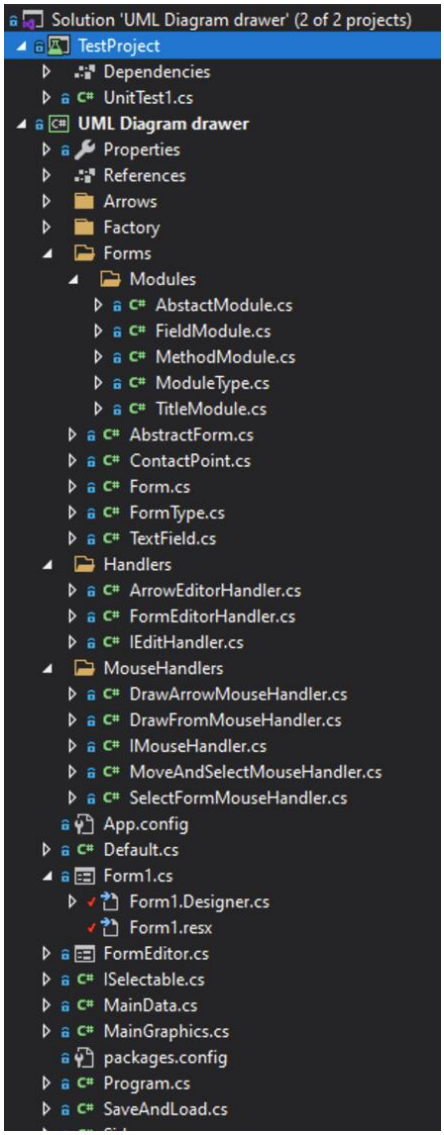

Рис. 2.6. Перелік файлів проекту у вікні оглядача рішень у Microsoft Visual Studio 2019

Для більш детального огляду функцій програми та варіанту їх використаня розглянемо Use Case діаграму додатку UML_Builder.

Use Case - це термін, присвячений програмного і системного проектування, який описує, як користувач використовує систему для досягнення певної мети. Даний процес діє як метод моделювання програмного забезпечення, що визначає функції, які повинні бути реалізовані, і вирішення будь-яких помилок, які можуть виникнути.

Існує три основних елементи процесу:

– "Актори" - це тип користувачів, які взаємодіють з системою;

– система - функціональні вимоги, які визначають передбачувану поведінку елементів.

Use Case діаграми є цінними для візуалізації функціональних вимог системи, які будуть втілюватися у виборі дизайну і пріоритності розвитку. Вони також допомагають виявляти будь-які внутрішні чи зовнішні чинники, які можуть впливати на систему, і їх слід брати до уваги.

Use Case - діаграми прецедентів забезпечують хороший позасистемний аналіз високого рівня і вказують, як система взаємодіє з учасниками, не турбуючись про деталі реалізації цієї функціональності[8].

Іншими словами, різновид використання описує, «хто» і «що» може зробити з розглянутою системою. Методика різновидів використання застосовується для виявлення вимог до поведінки системи, відомих також, як функціональні вимоги.

Діаграма сценаріїв використання програми для створення аналогу редактора UML- діаграм зображена на рис. 2.7.

Рис. 2.7. Діаграмма сценаріїв використання програми для створення UML-діаграм

В додатку для створення та редагування UML-діаграм використовуються дві основні форми:

– MenuForm – для роботи з графічними елементами (рисунок 2.8);

– EditForm – для надання користувачеві можливості зміни класів, об’єктів та стрілки.

Розглянемо головну форму MenuForm (рис. 2.8).

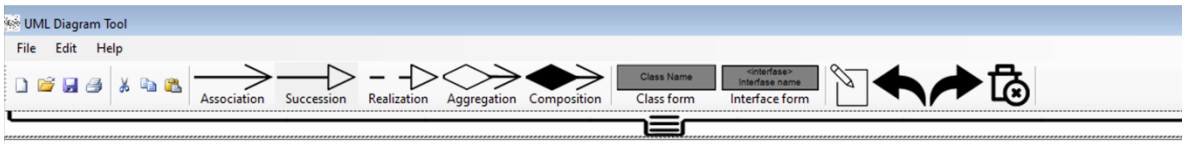

Рис. 2.8. Ескіз форми MenuForm в режимі конструктора

Видимими компонентами можна назвати ті, компоненти які розташовані на формі, такі як кнопки, текстові поля. Через наявність тіла на формі, можна редагувати їх розміри, колір, розташування, підказки, тощо.

Невидимі компоненти знаходяться поза формою, як окремі елементи, зазвичай вони не мають тіла у программі, але є виключення, як контексте меню. Серед цих елементів можна навести, контекстне меню, сховища данних, тощо. Через відсутність тіла, ці компоненти не можуть бути змінені фізічно, зазвичай при редагуванні, їх можна змінити логічно, на приклад у таймера інтервал.

Компоненти та події форми MenuForm наведині таблицях 2.1 та 2.2.

Таблиця 2.1.

Компоненти форми “Form”

Тип компоненту	Назва компоненту	Призначення
MenuStrip	MenuStrip1	Містить меню для роботи з системою проекту та виклику форм тощо
StatusStrip	StatusStrip1	Рядок стану
toolStrip	toolStrip1	Панель інструментів з усім функціаналом
pictureBoxHamburger	pictureBoxHamburger1	Зображення для закаріття toolStrip1
Button	FormMain_Load	Кнопка для завантаження UML-діаграм
	toolStripButtonEditObject	Кнопка для видалення об’єктів
	toolStripButtonSelectForm	Кнопка для виділення об’єктів
	toolStripButtonNewFile	Кнопка для відкрітя нового файлу
	toolStripButtonUndo	Кнопка для поверненя останньої дії
	toolStripButtonRedo	Кнопка для відміни дії
	toolStripButtonDelet	Кнопка для видалення об’єкту
	copyToStackButton	Кнопка для копіювання об’єкту
	PasteObject	Кнопка для вставки об’єкта

Таблиця 2.2.

Події форми “Form”

Подія	Призначення події
private void pictureBoxHamburger_Click(object sender, EventArgs e)	Подія для створення зображеняя
private void flowLayoutPanel1_Scroll(object sender, ScrollEventArgs e)	Подія для створеня скролів
private void EditObject_MouseDown(object sender, MouseEventArgs e)	Подія для видаляє об’єкта з панелі
private void PasteObject_MouseDown(object sender, MouseEventArgs e)	Вставляє об’єкта на відпускання миші
private void copyToStackButton_Click(object sender, EventArgs e)	Копіює об’єкта на відпускання миші
private void toolStripButtonRedo_Click(object sender, EventArgs e)	Відміняє останю дію на натисканні на кнопку
private void toolStripButtonNewFile_Click(object sender, EventArgs e)	Зберігає файл
private void toolStripButtonSelectForm_Click(object sender, EventArgs e)	Вибирає об’єкта
private void toolStripButtonEditObject_Click(object sender, EventArgs e)	Відкріває FormEditor
private void FormMain_Load(object sender, EventArgs e)	Завантажує файл

Оскільки форма для створення UML-діаграм є єдиною формою з можливістю використання всіх функцій програми, то вона повинна з’являтися одразу після запуску програми. Ця форма зібрала в собі великий склад компонентів та подій. Вона містить головні функції у швидкому доступі (гарячі клавіші) та вспливаючі підказки для зрозумілості користувачам. За допомогою головного меню можна обирати вкладки для зручності редагування діаграми, вони розподілені за певним функціоналом, а саме робота з файлами, об’єктами та допомога.

Форма “EditForm” надає користувачеві можливість редагувати класи, інтерфейси та стрілки, ескіз форми наведений на рис. 2.9.

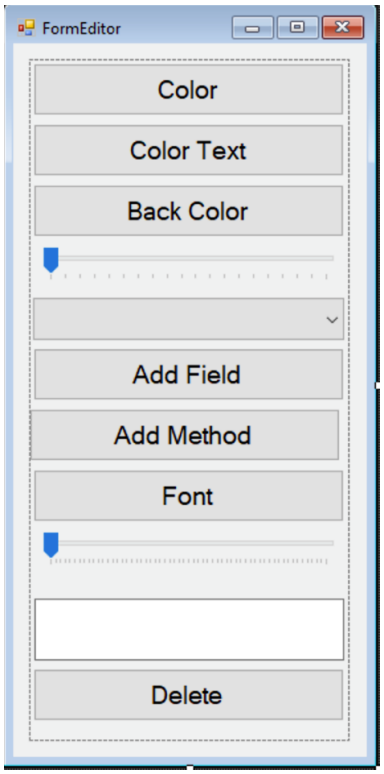

Рис. 2.9. Ескіз форми EditForm в режимі конструктора

Компоненти та події форми “EditForm” наведені у таблицях 2.3 та 2.4.

Таблиця 2.3.

Компоненти форми “EditForm”

Типкомпоненту	Назва компоненту	Призначення
button	buttonColorChoice	Кнопка вибору коляру
	buttonSelectFont	Кнопка вибору шрифту
	buttonAddMethod	Кнопка для додаваня методу
	buttonAddField_Click	Кнопка для додаваня поля
	buttonColorTextChoice	Кнопка вибору коляру букв
	buttonSetBackColor	Кнопка для вибору фону
	buttonDelete	Кнопка для видалення змін
trackBar	trackBarSizeForm	Скрол для вибору розміру
textBox	textBoxSelectTextField	Текстове поле для введеня даних

Таблиця 2.4.

Події форми “EditForm”

Подія	Призначення події
private void FormEditor_Load(object sender, EventArgs e)	Подія відкриття форми
private void buttonColorChoice_Click(object sender, EventArgs e)	Зміна фону
private void trackBarSizeForm_Scroll(object sender, EventArgs e)	Зміна товщини грані
private void buttonSelectFont_Click(object sender, EventArgs e)	Зміна шрифту
private void buttonAddField_Click(object sender, EventArgs e)	Додавання поля
private void buttonAddMethod_Click(object sender, EventArgs e)	Додавання методу
private void buttonColorTextChoice_Click(object sender, EventArgs e)	Зміна фону текста
private void trackBarLineThickness_Scroll(object sender, EventArgs e)	Розмір шрифта по скролу
private void comboBoxSetTypeArrow_SelectedIndexChanged(object sender, EventArgs e)	Вибір типу стрілки

Діаграма класів – статичне представлення структури моделі. Відображає статичні (декларативні) елементи, такі як: класи, типи даних, їх зміст та відношення. Діаграма класів, також, може містити позначення для пакетів та може містити позначення для вкладених пакетів. Також, діаграма класів може містити позначення деяких елементів поведінки, однак їх динаміка розкривається в інших типах діаграм[9].

Діаграма класів служить для представлення статичної структури моделі системи в термінології класів об'єктно-орієнтованого програмування. На цій діаграмі показують класи, інтерфейси, об'єкти й кооперації, а також їхні відносини [9].

На діаграммі мають розташовані всі класи додатку, кожен з них має в собі велику кількість функцій та компонентів. Класи відповидають за збереження та обробку інформації певними методами, які належать певному класу.

2.5. Обгрунтування та організація вхідних та вихідних даних програми

У якості вхідних даних виступають: класи, інтерфейси і зв’язки між ними (UML-діаграм).

Вихіднимі даними є готова UML-діаграма збережена в файли формату JPEG або (umld)UML-діаграм.

2.6. Опис роботи розробленої системи

2.6.1. Використані технічні засоби

При виконанні кваліфікаційної роботи та розробці додатку було використано ноутбук ASUS TUF Gaming F15 Graphite Black / 15.6" IPS / Intel Core i5-11400H / RAM 8 ГБ / SSD 512 ГБ / nVidia GeForce RTX 3050.

2.6.2. Використані програмні засоби

Для розробки програмного додатку була обрана мова програмування C# та середовище розробки Microsoft Visual Studio 2019.

2.6.3. Виклик та завантаження програми

Для початку роботи з програмою UML_Builder необхідно запустити на виконання файл UML_builder.exe.

2.6.4. Опис інтерфейсу користувача

Після запуску файлу програми перед користувачем відкриється головне вікно програми, яке дозволяє розпочати роботу по створенню діаграм (рис.2.10).

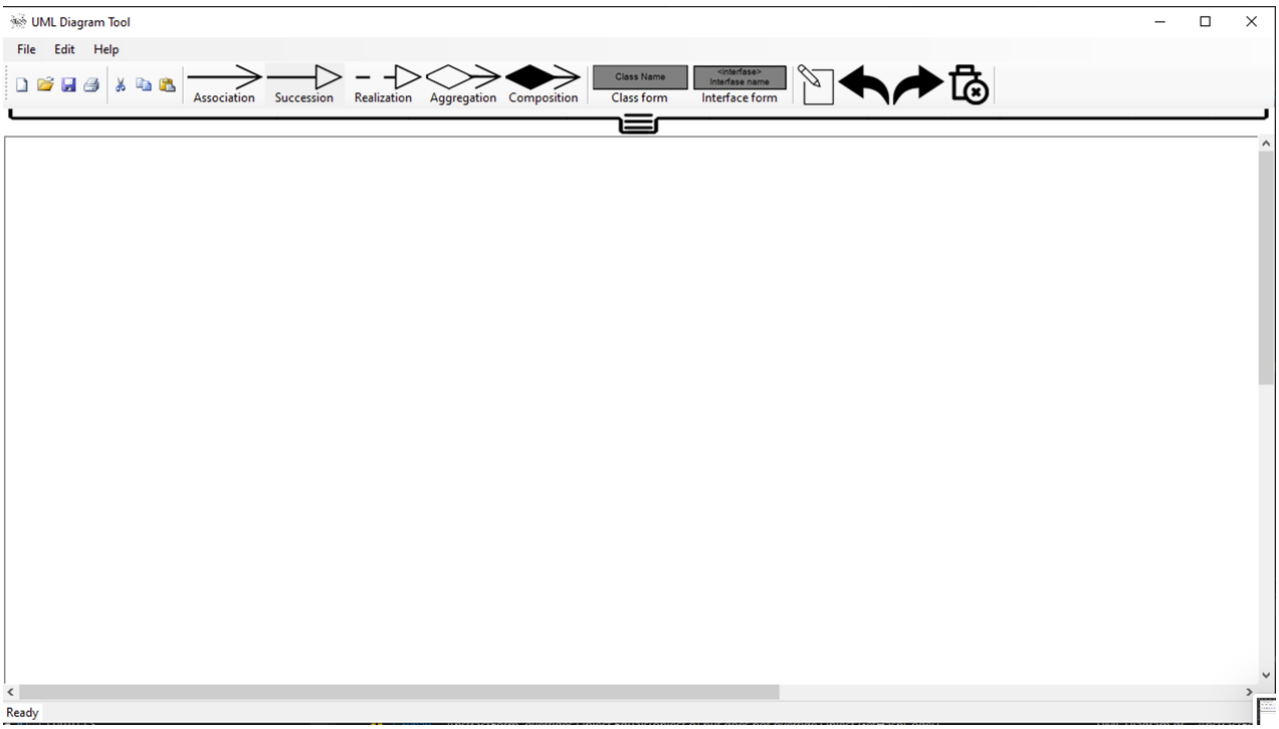

Рис. 2.10. Головне вікно програми UML_Builder

Головне вікно програми складається з:

– заголовку програми, де міститься системний ярлик програми, назва програми та кнопок керування вікном;

– головного меню;

– панелі інструментів з усім функціоналом програми;

– рядку стану, який містить в собі назву поточного режиму;

– вертикального і горизонтального скролів.

Головне меню програми складається з трьох пунктів, а саме: файл, редагування, довідка. Перший пункт – файл, дозволяє працювати з файлами, що створити, відкрити, зберегти у форматі Jpeg або у форматі UML диаграма готову UML-диаграму. Далі іде пункт редагування, де можна скопіювати, вставити, вирізати або редагувати виділенний об’єкт, класів, інтерфейсів та стрілок. Третій пункт складаеться з усіх елементів для побудови UML-диаграм, а саме класси, інтерфейси та різні види стрілок.

Процес роботи з UML-діаграмою в данній програмі поділяється на два типи, це робота над вже готовою або недопрацьованою UML-диаграмою, або розробка нової UML-діаграми.

В випадку розробки нової UML-диаграми для початку користувач повинен розмістити на панелі усі класи та інтерфейси , це можна виконати зявдяки кнопкам 

.

Після розміщення класів та інтерфейсів користувачу потрібно встановити необхідні звязки між цими класами та інтерфейсами, використовуючи відповідні кнопки на панелі інструментів 

(рис. 2.11).

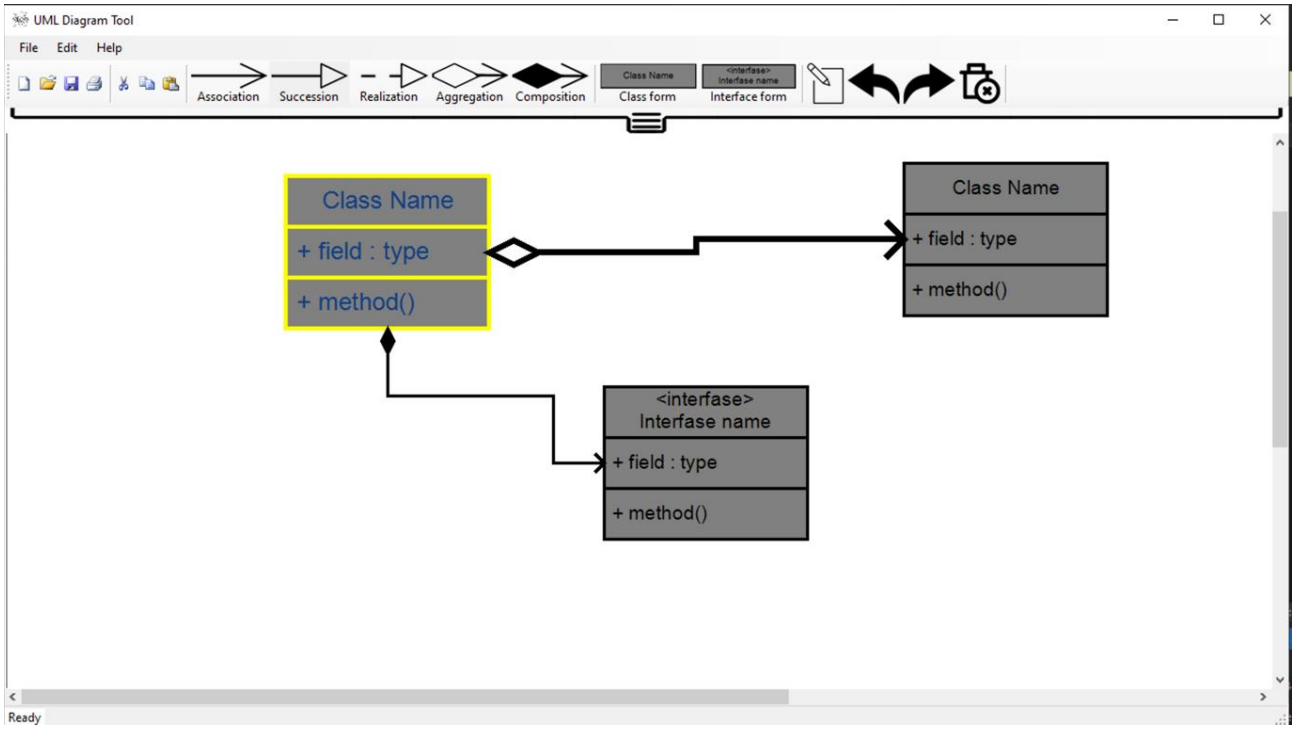

Рис. 2.11. Головне вікно програми зі створеною UML-диаграмою

При розробці UML-диаграми користувачу може стати замало данного поля, він може скористатися скролами, які роблять полотно більшим при його переміщенні. Полотно є нескінченно великим.

Під час роботи з UML-диаграмою користувач має можливість відмініти останню дію, за допомогою кнопок ”Назад” та ”Вперед”  на панелі інструментів.

Для редагування об’єктів на формі використовується кнопка Edit  на панелі інструментів. Для цього необхідно спочатку обрати об’єкт для редагування, натиснувши на нього правою кнопкою миші, потім після натискання на кнопку Edit на панелі інструментів, стає доступним вікно для внесення відповідних змін, але для кожного об’єкту воно різне. Для стрілочки воно містить ширину, колір та меню типу стрілки (рис. 2.12).

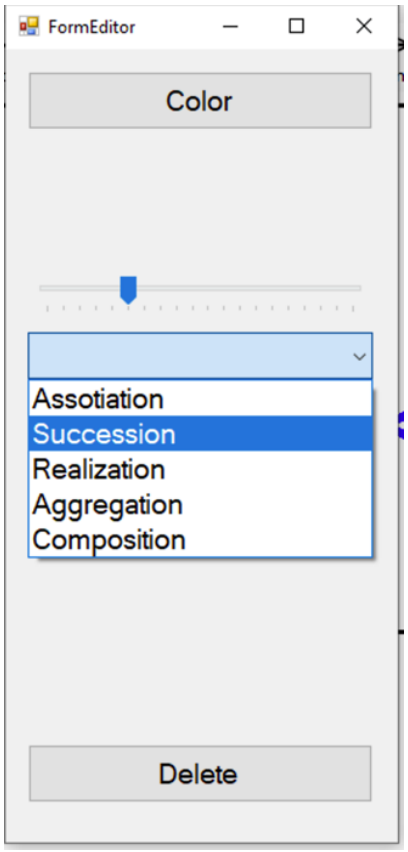

Рис. 2.12. Допоміжне вікно для редагування стрілок програми UML_Builder

Для створення зв’язків між об’єктами у програмі передбачаються стрілки. Типи стрілок, які може обрати користувач:

– суцільна;

– пунктирна;

– штрихована;

– штрих-пунктирна;

– штрих-штрих-пунктирна.

Товщина ліній стрілок змінюється за допомогою повзунка від 1 до 10. Для вибору кольору стрілки необхідно натиснути на кнопку Color і відкривається діалогове вікно для зміни коляру.

Також користувач може редагувати класи або інтерфейси, а саме текст, розмір шрифту, його стиль, колір рамки або додати більше полів тощо (рис.2.13).

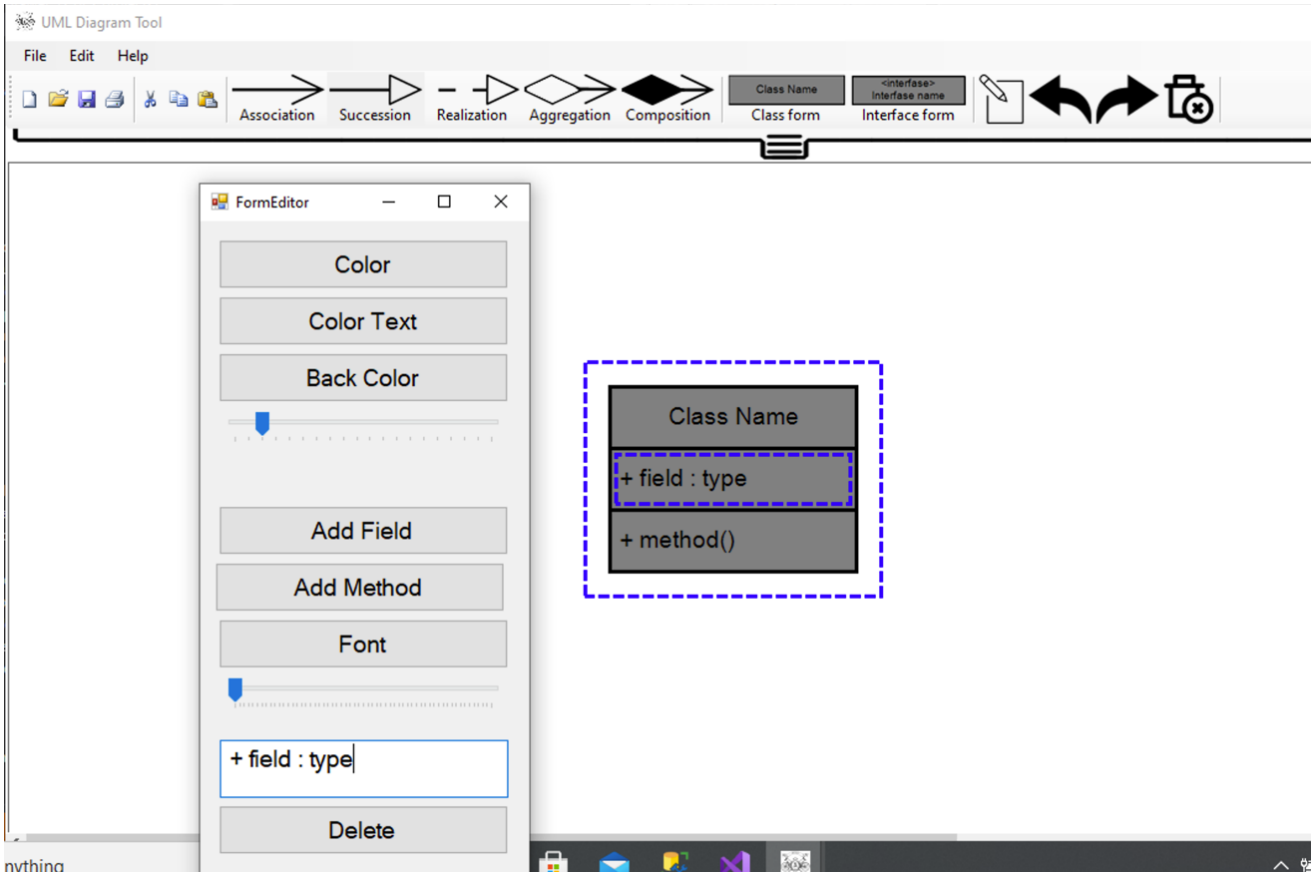

Рис. 2.13. Діалогове вікно FormEditor для форматування форми.

Також користувач може вибрати будь який об’єкти і скопіювати, вставити або видалити об’єкти  , або використовуючи гарячі клавіши Ctrl+C, Ctrl+V,Ctrl +E аналогічно.

Після завершення роботи з діаграмою необхідно зберегти. Для цього можна обрати в головному меню “Файл”->”Зберегти” або “Зберегти у форматі JPEG”, або за допомогою гарячих клавіш Ctrl+S або Ctrl+J (рис. 2.14).

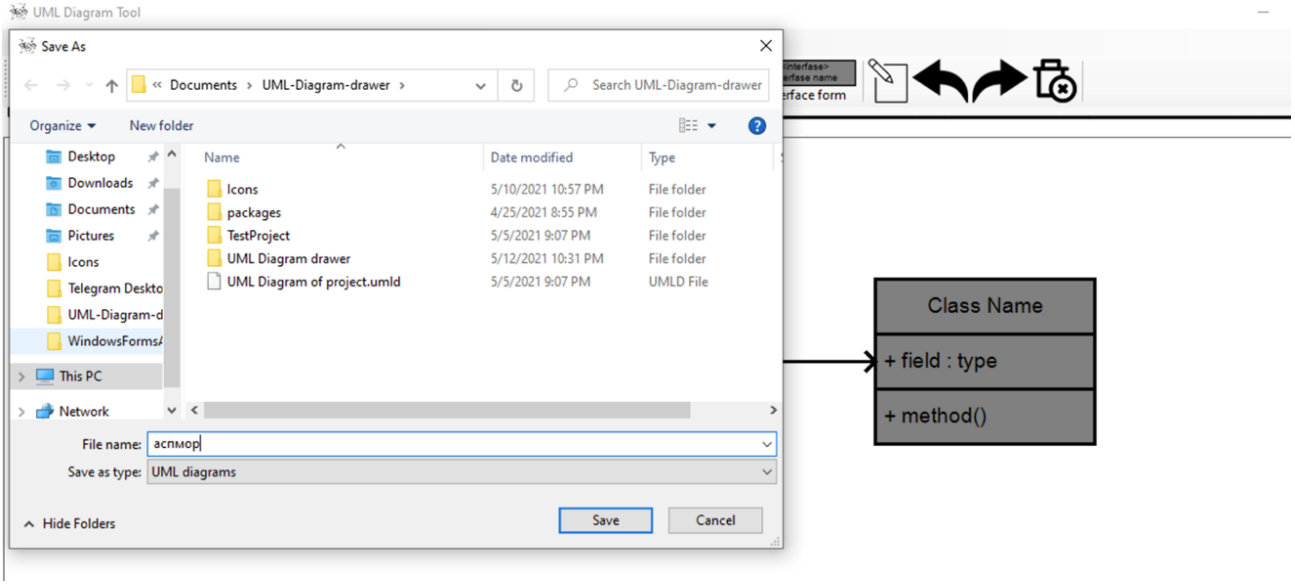

Рис. 2.14. Діалогове вікно для збереження UML-диаграми

Після збереження результат роботи, а саме UML-диаграми, буде зберігатися у файлі обраного формату, JPEG або UML-діаграм, у обраному користувачем каталозі.

Для редагування UML-диаграми, необхідно її спочатку відкрити, натиснувши на кнопку open file на головному менб додатка і відкрити потрібний файл (рис. 2.15).

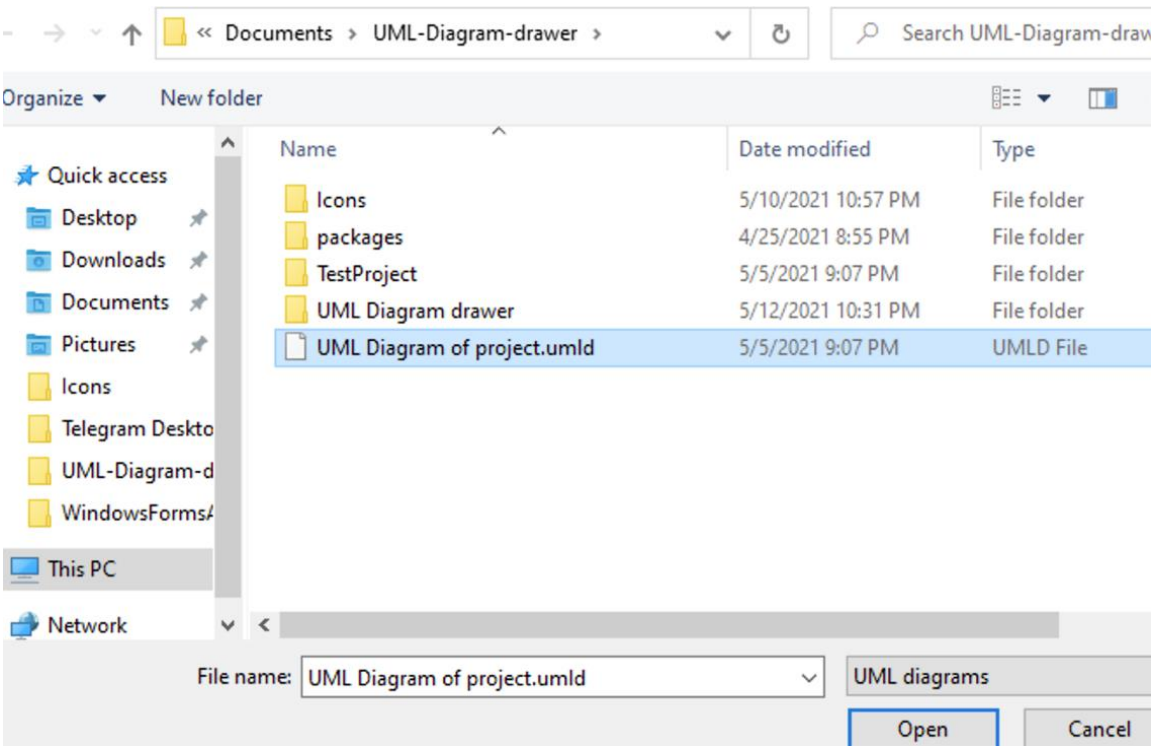

Рис. 2.15. Вікно для відкриття збереженого файлу

В іншому випадку, коли UML-діаграма вже була спроектована і засобами додатку потрібно її відредагувати, для цього користувачу потрібно завантажити файл формату UML-диаграм у програму.

Завантаження здійснюється за домогою пункту головного меню File -> Open,

або за допомогою кнопки на панелі інструментів або комбінація гарячої клавіши Ctrl+O. Після натискання на пункт, кнопку або комбінацію гарячої клавішу відкривається діалогове вікно для пошуку файлу, користувачу слід обрати файл і натиснути клавішу Відкрити.

Після відкриття UML-диаграми користувачу потрібно проаналізувати попередню роботу, та зрозуміти що потрібно доопрацювати, або що потрібно відалити зайві, або редагувати класи, інтерфейси, зв’язки.

Для перегляду без редагування UML-диаграми потрібно закрити усі інструменти, це здійснюється за допомогою натискання на кнопку

 .

Після завершеня роботи результат потрібно зберегти (рис. 2.16).

Слід зазначити, що програмний продукт містить підказки для більшого розуміння користувача. На останій вкладці меню розміщений пункт меню Help, який містить в собі вікно зі списком гарячих клавіш і списоком поширених питань (рис. 2.17 - 2.18).

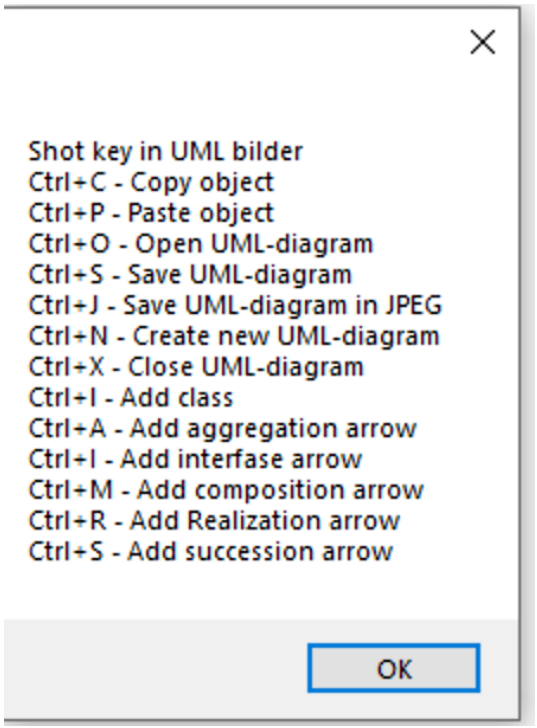

Рис. 2.17. Вікно з комбінаціями гарячих клавіш

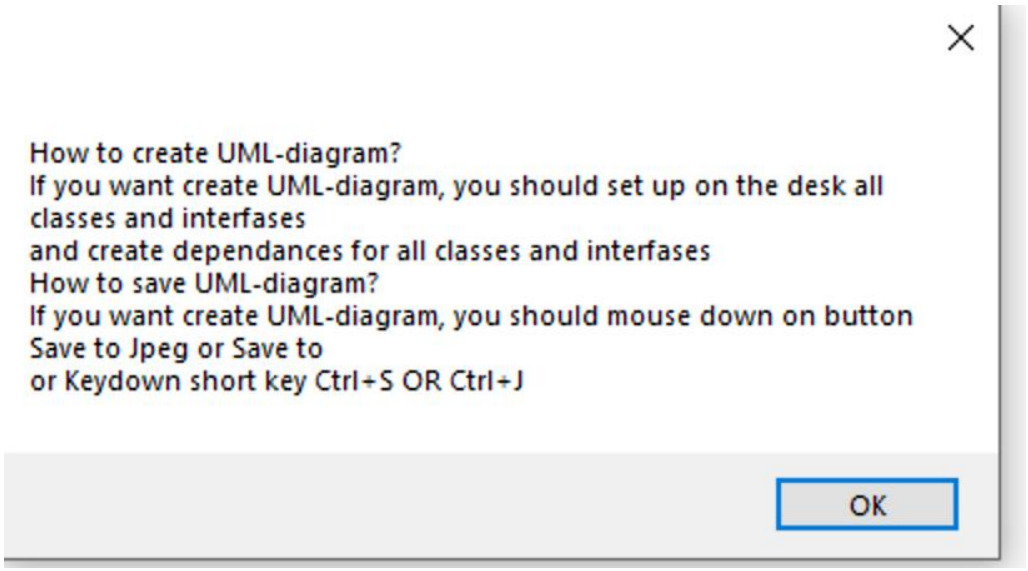

Рис. 2.18. Вікно поширених питань

Для перевірки програми на стабільність роботи та виключення помилок у коді, програма повинна пройти модульне тестування[10].

Тестування програмного продукту відбувалося за допомогою вбудованих систем в Visual Studio 2019 NUnit тестов. Програмний продукт пройшов тестування без помилок, а це означає що він є придатним для роботи. Тестування проходило над функціями та елементами програми, як видимими, так і не видимими усіх public методів (рис. 2.19).

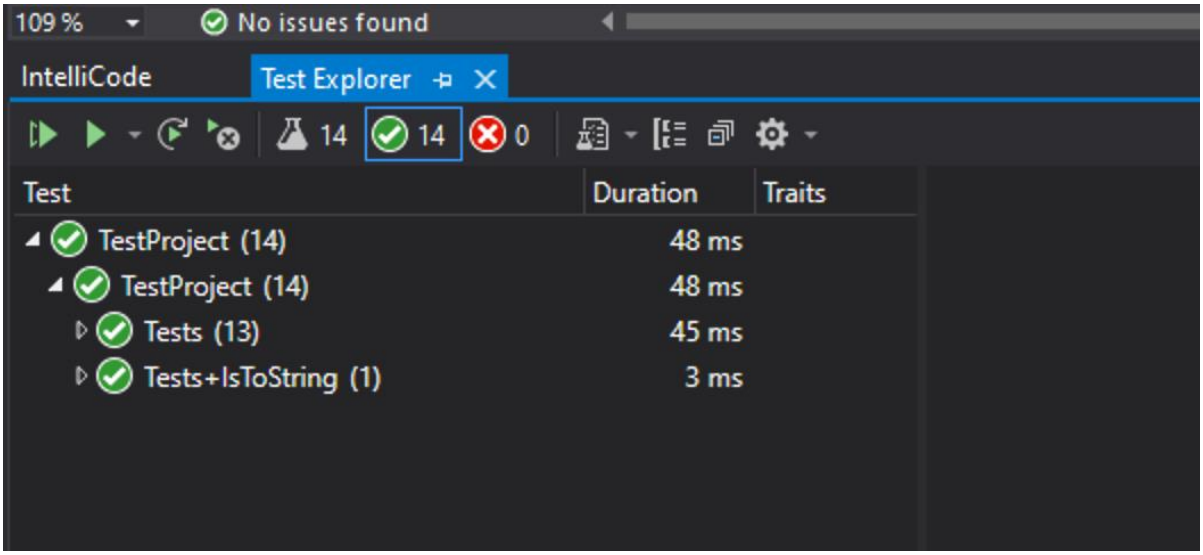

Рис. 2.19. Тестування програмного продукту

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Початкові дані:

1. передбачуване число операторів програми – 1200;
2. коефіцієнт складності програми – 1,6;
3. коефіцієнт корекції програми в ході її розробки – 0,05;
4. годинна заробітна плата програміста – 60 грн/год;
5. коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,2;
6. коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1,2;
7. вартість машино-години ЕОМ – 13 грн/год.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д \text{, людино-годин,} \quad (3.1)$$

де t_o - витрати праці на підготовку й опис поставленої задачі (приймається 50 людино-годин);

t_u - витрати праці на дослідження алгоритму рішення задачі;

t_a - витрати праці на розробку блок-схеми алгоритму;

t_n - витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ - витрати праці на налагодження програми на ЕОМ;

$t_д$ - витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у програмногму забезпеченні, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C \cdot (1 + p),$$

де q - передбачуване число операторів (1200);

C - коефіцієнт складності програми (1,6);

p - коефіцієнт корекції програми в ході її розробки (0,05).

Звідси умовне число операторів в програмі:

$$Q = 1{,}6 \cdot 1200 \cdot (1 + 0{,}05) = 2016$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75..85) \cdot k}, \text{ людино-годин,}$$

де B - коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

k - коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності. При стажі роботи від 3 до 5 років він складає 1,2.

Приймемо збільшення витрат праці внаслідок недостатнього опису завдання не більше 50% (B = 1.2). З урахуванням коефіцієнта кваліфікації k = 1,2, отримуємо витрати праці на вивчення опису завдання:

$$t_u = (2016 \cdot 1{,}2) / (75 \cdot 1{,}2) = 26{,}88 \text{ людино-годин}$$

Витрати праці на розробку алгоритму рішення задачі визначаються за формулою:

$$t_a = \frac{Q}{(20\ldots25)\cdot k}\ \text{, людино-годин,}\quad (3.2)$$

де Q – умовне число операторів програми;

k – коефіцієнт кваліфікації програміста.

Підставивши відповідні значення в формулу (3.2), отримаємо:

$$t_a = 2016 / (20 \cdot 1{,}2) = 84 \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20\ldots25)\cdot k}\ \text{, людино-годин.}$$

$$t_n = 2016 / (25 \cdot 1{,}2) = 67{,}2 \text{ людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ:

- за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4..5)\cdot k}\ \text{, людино-годин.}$$

$$t_{отл} = 2016 / (5 \cdot 1{,}2) = 336 \text{ чел.-ч.}$$

- за умови комплексного налагодження завдання:

$$t_{отл}^{k} = 1{,}5 \cdot t_{отл}\ \text{, людино-годин.}$$

$$t^{k}_{отл} = 1,5 \cdot 336 = 504 \text{ людино-годин.}$$

Витрати праці на підготовку документації визначаються за формулою:

$$t_{д} = t_{др} + t_{до} \text{ , людино-годин,}$$

де $t_{др}$ - трудомісткість підготовки матеріалів і рукопису:

$$t_{др} = \frac{Q}{(15..20) \cdot k} \text{ , людино-годин,}$$

$t_{до}$ - трудомісткість редагування, печатки й оформлення документації:

$$t_{до} = 0,75 \cdot t_{др} \text{ , людино-годин.}$$

Підставляючи відповідні значення, отримаємо:

$$t_{др} = 2016 / (18 \cdot 1,2) = 93,33 \text{ людино-годин.}$$

$$t_{до} = 0,75 \cdot 93,33 = 70 \text{ людино-годин.}$$

$$t_{д} = 93,33 + 70 = 163,33 \text{ людино-годин.}$$

Повертаючись до формули (3.1), отримаємо повну оцінку трудомісткості розробки програмного забезпечення:

$$t = 50 + 26,88 + 84 + 67,2 + 336 + 163,33 = 727,41 \text{ людино-годин.}$$

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ $К_{ПО}$ включають витрати на заробітну плату виконавця програми $З_{ЗП}$ і витрат машинного часу, необхідного на налагодження програми на ЕОМ:

$$К_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн.}$$

Заробітна плата виконавців визначається за формулою:

$$З_{ЗП} = t \cdot С_{ПР}, \text{ грн,}$$

де: t - загальна трудомісткість, людино-годин;

$С_{ПР}$ - середня годинна заробітна плата програміста, грн/година

З урахуванням того, що середня годинна зарплата програміста становить 60 грн / год, отримуємо:

$$З_{ЗП} = 727{,}41 \cdot 60 = 43\,644{,}6 \text{ грн.}$$

Вартість машинного часу, необхідного для налагодження програми на ЕОМ, визначається за формулою:

$$З_{мв} = t_{отл} \cdot С_{мч}, \text{ грн,} \quad (3.3)$$

де $t_{отл}$ - трудомісткість налагодження програми на ЕОМ, год;

$С_{мч}$ - вартість машино-години ЕОМ, грн/год (13 грн/год).

Підставивши в формулу (3.3) відповідні значення, визначимо вартість необхідного для налагодження машинного часу:

$$З_{мв} = 336 \cdot 13 = 4368 \text{ грн.}$$

Звідси витрати на створення програмного продукту:

$$К_{ПО} = 43644{,}6 + 4368 = 48\,012{,}6 \text{ грн.}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p}, \text{ міс.}$$

де B_k - число виконавців (дорівнює 1);

F_p - місячний фонд робочого часу (при 40 годинному робочому тижні F_p=176 годин).

Звідси витрати на створення програмного продукту:

$$T = 727{,}41 / 1 \cdot 176 \approx 4 \text{ міс.}$$

Висновок. Основне функціональне призначеня графічного редактора, що розробляється в роботі це створення та редагування UML-діаграм. Вартість даного програмного забезпечення становить 48 тис. грн. і не вимагає додаткових витрат як при розробці програми. Очікуваний час розробки становить 4 місяці. Цей термін пов'язаний зі значним числом операторів, і включає час на дослідження і розробку алгоритму вирішення поставленого завдання, програмування по готовому алгоритму, налагодження програми і підготовку документації.

ВИСНОВКИ

Під час роботи над кваліфікаційною роботою була створена програма “UML editor“. Розроблена програма має великий функціонал та може використовуватися, як дома для персональних потреб, так і на комерційних підприємствах. У порівнянні з іншими програмними додатками, розроблена програма має більший та зручніший функціонал, а саме:

- не обмеженне поле для редагування;
- інтуітивно зрозумілий інтерфейс;
- можливість зміни розміру фігур та переміщення їх по полю;
- можливість збереження у форматі JPEG.

Всі поставлені в роботі завдання виконано в повному обсязі.

Програмний продукт пройшов тестування сторонніми програмами, які не виявили помилок у роботі та результат тестування був відмінним.

Також у дипломному проекті було визначено трудомісткість розробленої підсистеми (727 люд-год), проведений підрахунок вартості роботи по створенню програми (48012 грн) та розраховано час на його створення (4 міс). Данна програма може бути розповсюдженна в інтернеті.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Албахарі Джозеф. C# 3.0. лінки / Джозеф Албахарі, Бен Албахарі., 2013. - 944 с.
2. Альфред В. Ахо. Компілятори Принципи, технології та інструментарій / Альфред В. Ахо та ін., 2015. – 266 с.
3. Бішоп Дж. С # у короткому викладі / Дж. Бішоп, Н. Хорспул., 2013. – 472 с.
4. Вагнер Білл. C# Ефективне програмування / Білл Вагнер, 2013. – 320с.
5. Зіборов Віктор. Visual C# 2010 на прикладах / Віктор Зіборов., 2011. - 432 с.
6. Ішкова Е. А. Самовчитель С #. Початки програмування/Е.А. Ішкова., 2013. – 496 с.
7. Лотка Рокфорд. C# та CSLA .NET Framework. Розробка бізнес-об'єктів / Рокфорд Лотка., 2010. - 816 с.
8. Мак-Дональд Метью. Silverlight 5 з прикладами на C# для професіоналів / Метью МакДональд., 2013. - 848 с.
9. Подбельський В. В. Мова С #. Базовий курс/В.В. Ірпінь., 2011. – 384с.
10. Ріхтер Джеффрі. CLR за допомогою C#. Програмування на платформі Microsoft .NET Framework 4.0 мовою C# / Джеффрі Ріхтер., 2013. – 928 с.
11. Троєлсен Ендрю. Мова програмування C# 5.0 та платформа .NET 4.5 / Ендрю Троелсен., 2015. - 486 с.
12. Головний сайт з продукту Microsoft Visual Studio та описом продукту https://visualstudio.microsoft.com
13. Розгорнене створення та керування графікою за допомогою використання WinForms. URL: https://docs.microsoft.com/ ru/dotnet/framework/winforms/advanced/how-to-create-graphics-objects-for-drawing
14. Еве Немет Linux системне адміністування, Київ - 2012, - 203 с.
15. Графічний растровий редактор draw.it UML-діаграми, URL: https://www.pinterest.com/dana_sangster/draw-it/

16. Що таке Use Case діаграма https://dou.ua/lenta/articles/use-cases/
17. Діаграмма класів Вікіпедія https://uk.wikipedia.org/wiki/
18. Модульне тестування https://uk.wikipedia.org/wiki/

ДОДАТОК А

КОД ПРОГРАМИ

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace UML_Diagram_drawer
{
   static class Program
   {
      /// <summary>
      /// The main entry point for the application.
      /// </summary>
      [STAThread]
      static void Main()
      {
         Application.EnableVisualStyles();
         Application.SetCompatibleTextRenderingDefault(false);
         Application.Run(new FormMain());
      }
   }
}
```

```
using System;
using System.Collections.Generic;
using System.Drawing;
using Newtonsoft.Json;
using System.Windows.Forms;
using UML_Diagram_drawer.Forms;
using UML_Diagram_drawer.Arrows;
using Form = System.Windows.Forms.Form;
using UML_Diagram_drawer.Factory;
using UML_Diagram_drawer.MouseHandlers;
using UML_Diagram_drawer.Factory.ArrowFactories;

namespace UML_Diagram_drawer
{
   public partial class FormMain : Form
   {
      private MainData _mainData;

      public Pen pen = new Pen(Brushes.Black, 3);

      public FormMain()
      {
         InitializeComponent();
      }
```

```
#region Json
private string JsonSerialize(TypeOfData type)
{
    if (type == TypeOfData.Forms)
    {
        string fileDataForms = JsonConvert.SerializeObject(_mainData.FormsList, Formatting.Indented,
            new JsonSerializerSettings
            {
                TypeNameHandling = TypeNameHandling.All
            });

        return fileDataForms;
    }
    else if (type == TypeOfData.Arrows)
    {
        string fileDataArrows = JsonConvert.SerializeObject(_mainData.ArrowsList, Formatting.Indented,
            new JsonSerializerSettings
            {
                TypeNameHandling = TypeNameHandling.All
            });

        return fileDataArrows;
    }
    throw new Exception();
}

private void JsonDeserialize(string[] fileData)
{
    if (fileData[0] != String.Empty)
    {
        _mainData.FormsList = JsonConvert.DeserializeObject<List<AbstractForm>>(fileData[0],
            new JsonSerializerSettings
            {
                TypeNameHandling = TypeNameHandling.All
            });
    }
    else
    {
        _mainData.FormsList = new List<AbstractForm>();
    }

    if (fileData[1] != String.Empty)
    {

        _mainData.ArrowsList = JsonConvert.DeserializeObject<List<Arrow>>(fileData[1],
            new JsonSerializerSettings
            {
                TypeNameHandling = TypeNameHandling.All
            });
        foreach (var item in _mainData.ArrowsList)
        {
            item.Color = Color.Black;
        }
    }
    else
    {
        _mainData.ArrowsList = new List<Arrow>();
    }
}

#endregion
```

```
        private void RemoveArrowConnections()
        {
            foreach (Arrow arrow in _mainData.ArrowsList)
            {
                foreach (AbstractForm form in _mainData.FormsList)
                {
                    foreach (ContactPoint point in form.ContactPoints)
                    {
                        if (arrow.StartPoint.Equals(point))
                        {
                            arrow.StartPoint = point;
                        }
                        else
                        {
                            arrow.StartPoint.Location = Point.Empty;
                        }
                        if (arrow.EndPoint.Equals(point))
                        {
                            arrow.EndPoint = point;
                        }
                        else
                        {
                            arrow.EndPoint.Location = Point.Empty;
                        }
                    }
                }
            }

            for (int i = 0; i < _mainData.ArrowsList.Count; i++)
            {
                if (_mainData.ArrowsList[i].StartPoint.Location == Point.Empty ||
_mainData.ArrowsList[i].EndPoint.Location == Point.Empty)
                {
                    _mainData.ArrowsList.Remove(_mainData.ArrowsList[i]);
                    --i;
                }
            }
        }

        private void RebindingArrows()
        {
            foreach (Arrow arrow in _mainData.ArrowsList)
            {
                foreach (AbstractForm form in _mainData.FormsList)
                {
                    foreach (ContactPoint point in form.ContactPoints)
                    {
                        if (arrow.StartPoint.Equals(point))
                        {
                            arrow.StartPoint = point;
                        }
                        //else
                        //{
                        //    arrow.StartPoint.Location = Point.Empty;
                        //}
                        else if (arrow.EndPoint.Equals(point))
                        {
                            arrow.EndPoint = point;
                        }
                        //else
                        //{
                        //    arrow.EndPoint.Location = Point.Empty;
                        //}
```

```
                }
            }
        }

        //for (int i = 0; i < _mainData.ArrowsList.Count; i++)
        //{
        //    if (_mainData.ArrowsList[i].StartPoint.Location == Point.Empty||
_mainData.ArrowsList[i].EndPoint.Location == Point.Empty)
        //    {
        //        _mainData.ArrowsList.Remove(_mainData.ArrowsList[i]);
        //        --i;
        //    }
        //}
    }

    private void FormMain_Load(object sender, EventArgs e)
    {
        _mainData = MainData.GetMainData();
        _mainData.PictureBoxMain = pictureBoxMain;
        _mainData.FormsList = new List<AbstractForm>();
        _mainData.ArrowsList = new List<Arrow>();
        _mainData._formFactory = Default.Factory.Form;
        _mainData.IMouseHandler = new MoveAndSelectMouseHandler();
        _mainData.SaveChanges();
    }

    #region Tool Strip

    #region CreateArrows
    private void toolStripButtonArrowAssociation_Click(object sender, EventArgs e)
    {
        statusStripStatusLabel.Text = "Click to start point and drag mouse to end point, to crate arrow
association";
        _mainData._arrowsFactory = new ArrowAssociationFactory();
        _mainData.IMouseHandler = new DrawArrowMouseHandler();
    }
    private void toolStripButtonArrowSuccession_Click(object sender, EventArgs e)
    {
        _mainData._arrowsFactory = new ArrowSuccessionFactory();
        _mainData.IMouseHandler = new DrawArrowMouseHandler();
    }

    private void toolStripButtonArrowRealization_Click(object sender, EventArgs e)
    {
        _mainData._arrowsFactory = new ArrowRealizationFactory();
        _mainData.IMouseHandler = new DrawArrowMouseHandler();
    }

    private void toolStripButtonArrowAggregation_Click(object sender, EventArgs e)
    {
        _mainData._arrowsFactory = new ArrowAggregationFactory();
        _mainData.IMouseHandler = new DrawArrowMouseHandler();
    }

    private void toolStripButtonArrowComposition_Click(object sender, EventArgs e)
    {
        _mainData._arrowsFactory = new ArrowCompositionFactory();
        _mainData.IMouseHandler = new DrawArrowMouseHandler();
    }

    #endregion

    #region CreateForm
```

```
        private void toolStripButtonCreateClassForm_Click(object sender, EventArgs e)
        {
            _mainData._formFactory = new ClassFormFactory();
            _mainData.IMouseHandler = new DrawFromMouseHandler();
        }

        private void toolStripButtonInterfaceForm_Click(object sender, EventArgs e)
        {
            _mainData._formFactory = new InterfaceFormFactory();
            _mainData.IMouseHandler = new DrawFromMouseHandler();
        }

        #endregion

        private void toolStripButtonEditObject_Click(object sender, EventArgs e)
        {
            FormEditor formEditor = new FormEditor();
            formEditor.Show();
            _mainData.IMouseHandler = new SelectFormMouseHandler();
        }

        private void toolStripButtonSelectForm_Click(object sender, EventArgs e)
        {
            _mainData.IMouseHandler = new SelectFormMouseHandler();
        }

        private void toolStripButtonNewFile_Click(object sender, EventArgs e)
        {
            if (_mainData.FormsList.Count != 0)
            {
                const string infoText = "Создание нового файла удалит несохраненные изменения. Сохранить
изменения?";
                const string warningText = "Внимание!";
                DialogResult dialogResult = MessageBox.Show(infoText, warningText,
MessageBoxButtons.YesNoCancel, MessageBoxIcon.Warning);
                if (dialogResult == DialogResult.Yes)
                {
                    toolStripButtonSaveFile_Click(sender, e);
                }
                else if (dialogResult == DialogResult.No)
                {
                    _mainData.FormsList = new List<AbstractForm>();
                    _mainData.ArrowsList = new List<Arrow>();
                    _mainData.PictureBoxMain.Invalidate();
                }
            }
        }

        #region Save&Load
        private void toolStripButtonSaveFile_Click(object sender, EventArgs e)
        {
            if (saveFileDialog1.ShowDialog() == DialogResult.OK)
            {
                string fileDataForms = JsonSerialize(TypeOfData.Forms);
                string fileDataArrows = JsonSerialize(TypeOfData.Arrows);
                SaveAndLoad.SaveFile(saveFileDialog1.FileName, fileDataForms, fileDataArrows);
            }
        }

        private void toolStripButtonOpenFile_Click(object sender, EventArgs e)
        {
            MessageBox.Show("Переустановить виндовс?", "Во халепа", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
```

```
            if (openFileDialog1.ShowDialog() == DialogResult.OK)
            {
                string[] fileData = SaveAndLoad.OpenFile(openFileDialog1.FileName);
                JsonDeserialize(fileData);
                RebindingArrows();
                _mainData.PictureBoxMain.Invalidate();
            }
        }

        private void toolStripButtonSaveImageFile_Click(object sender, EventArgs e)
        {
            if (saveFileDialog2.ShowDialog() == DialogResult.OK)
            {
                Bitmap bmp = new Bitmap(pictureBoxMain.Width, pictureBoxMain.Height);
                pictureBoxMain.DrawToBitmap(bmp, new Rectangle(0, 0, pictureBoxMain.Width,
pictureBoxMain.Height));
                bmp.Save(saveFileDialog2.FileName, System.Drawing.Imaging.ImageFormat.Jpeg);
            }
        }
        #endregion

        #region CopyPaste
        private void toolStripButtonCopy_Click(object sender, EventArgs e)
        {
            copyToStackButton_Click(sender, e);
        }

        private void toolStripButtonPaste_Click(object sender, EventArgs e)
        {
            pictureBoxMain.MouseDown += PasteObject_MouseDown;
        }

        private void toolStripButtonCut_Click(object sender, EventArgs e)
        {
            pictureBoxMain.MouseDown += EditObject_MouseDown;
        }

        #endregion

        private void toolStripButtonUndo_Click(object sender, EventArgs e)
        {
            MainData.UnDo();
            _mainData = MainData.GetMainData();
        }

        private void toolStripButtonRedo_Click(object sender, EventArgs e)
        {
            MainData.ReDo();
            _mainData = MainData.GetMainData();
        }

        private void toolStripButtonDelete_Click(object sender, EventArgs e)
        {
            if (_mainData.SelectArrow != null)
            {
                _mainData.ArrowsList.Remove(_mainData.SelectArrow);
                _mainData.SelectArrow = null;
            }
            else if (_mainData.SelectForm != null)
            {
                _mainData.FormsList.Remove(_mainData.SelectForm);
                _mainData.SelectForm = null;
                RemoveArrowConnections();
```

```
            }

            _mainData.PictureBoxMain.Invalidate();
        }

        #endregion

        private void copyToStackButton_Click(object sender, EventArgs e)
        {
            _mainData.FormInBuffer = null;

            foreach (AbstractForm form in _mainData.FormsList)
            {
                if (form.IsSelected)
                {
                    _mainData.FormInBuffer = new UML_Diagram_drawer.Forms.Form(form);
                }
            }
        }

        private void PasteObject_MouseDown(object sender, MouseEventArgs e)
        {
            if (e.Button == MouseButtons.Left)
            {
                if (_mainData.FormInBuffer != null)
                {
                    _mainData.FormInBuffer.Location = e.Location;
                    _mainData.FormsList.Add(_mainData.FormInBuffer);
                    _mainData.FormInBuffer = null;
                }
            }

            pictureBoxMain.MouseDown -= PasteObject_MouseDown;
            pictureBoxMain.Invalidate();
        }

        private void EditObject_MouseDown(object sender, MouseEventArgs e)
        {
            if (e.Button == MouseButtons.Left)
            {
                if (_mainData.CurrentFormUML != null)
                {
                    _mainData.FormInBuffer = new
UML_Diagram_drawer.Forms.Form(_mainData.CurrentFormUML);
                    _mainData.FormsList.Remove(_mainData.CurrentFormUML);
                }
            }

            pictureBoxMain.Invalidate();
        }

        #region PictureBoxEvent
        private void pictureBoxMain_MouseClick(object sender, MouseEventArgs e)
        {
            if (_mainData.IMouseHandler != null)
            {
                _mainData.IMouseHandler.MouseClick(sender, e);
            }
        }

        private void pictureBoxMain_MouseDown(object sender, MouseEventArgs e)
        {
            if (_mainData.IMouseHandler != null)
```

```
    {
        _mainData.IMouseHandler.MouseDown(sender, e);
    }
}

private void pictureBoxMain_MouseUp(object sender, MouseEventArgs e)
{
    if (_mainData.IMouseHandler != null)
    {
        _mainData.IMouseHandler.MouseUp(sender, e);
    }
    statusStripStatusLabel.Text = "Ready";
}

private void pictureBoxMain_MouseMove(object sender, MouseEventArgs e)
{
    if (_mainData.IMouseHandler != null)
    {
        _mainData.IMouseHandler.MouseMove(sender, e);
    }
}

private void PictureBoxMain_Paint(object sender, PaintEventArgs e)
{
    MainGraphics.Graphics = e.Graphics;

    foreach (AbstractForm form in _mainData.FormsList)
    {
        form.Draw();
    }

    foreach (var arrow in _mainData.ArrowsList)
    {
        arrow.Draw();
    }
}

#endregion

private void flowLayoutPanel1_Scroll(object sender, ScrollEventArgs e)
{
    if (e.ScrollOrientation == ScrollOrientation.VerticalScroll)
    {
        pictureBoxMain.Height += e.NewValue - e.OldValue;
    }
    else
    {
        pictureBoxMain.Width += e.NewValue - e.OldValue;
    }
}

private void pictureBoxHamburger_Click(object sender, EventArgs e)
{
    if(toolStrip1.Visible==false)
    {
        toolStrip1.Visible = true;
        menuStrip1.Visible = true;
    }
    else if(toolStrip1.Visible==true)
    {
        toolStrip1.Visible = false;
        menuStrip1.Visible = false;
    }
```

```
        }

        private void menuStrip1_ItemClicked(object sender, ToolStripItemClickedEventArgs e)
        {

        }

        private void toolStrip1_ItemClicked(object sender, ToolStripItemClickedEventArgs e)
        {

        }

        private void createNewFileToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (_mainData.FormsList.Count != 0)
            {
                const string infoText = "Создание нового файла удалит несохраненные изменения. Сохранить
изменения?";
                const string warningText = "Внимание!";
                DialogResult dialogResult = MessageBox.Show(infoText, warningText,
MessageBoxButtons.YesNoCancel, MessageBoxIcon.Warning);
                if (dialogResult == DialogResult.Yes)
                {
                    toolStripButtonSaveFile_Click(sender, e);
                }
                else if (dialogResult == DialogResult.No)
                {
                    _mainData.FormsList = new List<AbstractForm>();
                    _mainData.ArrowsList = new List<Arrow>();
                    _mainData.PictureBoxMain.Invalidate();
                }
            }
        }

        private void saveFileInToJPEGToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (saveFileDialog2.ShowDialog() == DialogResult.OK)
            {
                Bitmap bmp = new Bitmap(pictureBoxMain.Width, pictureBoxMain.Height);
                pictureBoxMain.DrawToBitmap(bmp, new Rectangle(0, 0, pictureBoxMain.Width,
pictureBoxMain.Height));
                bmp.Save(saveFileDialog2.FileName, System.Drawing.Imaging.ImageFormat.Jpeg);
            }
        }

        private void cutObjectsToolStripMenuItem_Click(object sender, EventArgs e)
        {
            pictureBoxMain.MouseDown += EditObject_MouseDown;
        }

        private void interfaceToolStripMenuItem_Click(object sender, EventArgs e)
        {
            _mainData._formFactory = new InterfaceFormFactory();
            _mainData.IMouseHandler = new DrawFromMouseHandler();
        }

        private void classToolStripMenuItem_Click(object sender, EventArgs e)
        {
            _mainData._formFactory = new ClassFormFactory();
            _mainData.IMouseHandler = new DrawFromMouseHandler();
        }

        private void compositionToolStripMenuItem_Click(object sender, EventArgs e)
```

```
    {
        _mainData._arrowsFactory = new ArrowCompositionFactory();
        _mainData.IMouseHandler = new DrawArrowMouseHandler();
    }

    private void openFileToolStripMenuItem_Click(object sender, EventArgs e)
    {
        MessageBox.Show("Переустановить виндовс?", "Во халепа", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
        if (openFileDialog1.ShowDialog() == DialogResult.OK)
        {
            string[] fileData = SaveAndLoad.OpenFile(openFileDialog1.FileName);
            JsonDeserialize(fileData);
            RebindingArrows();
            _mainData.PictureBoxMain.Invalidate();
        }
    }

    private void FormMain_KeyDown(object sender, KeyEventArgs e)
    {
        if (e.Control && e.KeyCode == Keys.S)
        {
            if (saveFileDialog2.ShowDialog() == DialogResult.OK)
            {
                Bitmap bmp = new Bitmap(pictureBoxMain.Width, pictureBoxMain.Height);
                pictureBoxMain.DrawToBitmap(bmp, new Rectangle(0, 0, pictureBoxMain.Width,
pictureBoxMain.Height));
                bmp.Save(saveFileDialog2.FileName, System.Drawing.Imaging.ImageFormat.Jpeg);
            }
        }
        if (e.Control && e.KeyCode == Keys.O)
        {
            if (saveFileDialog1.ShowDialog() == DialogResult.OK)
            {
                string fileDataForms = JsonSerialize(TypeOfData.Forms);
                string fileDataArrows = JsonSerialize(TypeOfData.Arrows);
                SaveAndLoad.SaveFile(saveFileDialog1.FileName, fileDataForms, fileDataArrows);
            }
        }
        if (e.Control && e.KeyCode == Keys.J)
        {
            if (saveFileDialog2.ShowDialog() == DialogResult.OK)
            {
                Bitmap bmp = new Bitmap(pictureBoxMain.Width, pictureBoxMain.Height);
                pictureBoxMain.DrawToBitmap(bmp, new Rectangle(0, 0, pictureBoxMain.Width,
pictureBoxMain.Height));
                bmp.Save(saveFileDialog2.FileName, System.Drawing.Imaging.ImageFormat.Jpeg);
            }
        }
        if (e.Control && e.KeyCode == Keys.E)
        {


        }
        if (e.Control && e.KeyCode == Keys.C)
        {
            copyToStackButton_Click(sender, e);
        }
        if (e.Control && e.KeyCode == Keys.P)
        {
            pictureBoxMain.MouseDown += PasteObject_MouseDown;
        }
        if (e.Control && e.KeyCode == Keys.D)
```

```
        {
            if (_mainData.SelectArrow != null)
            {
                _mainData.ArrowsList.Remove(_mainData.SelectArrow);
                _mainData.SelectArrow = null;
            }
            else if (_mainData.SelectForm != null)
            {
                _mainData.FormsList.Remove(_mainData.SelectForm);
                _mainData.SelectForm = null;
                RemoveArrowConnections();
            }

            _mainData.PictureBoxMain.Invalidate();
        }

        if (e.Control && e.KeyCode == Keys.L)
        {
            _mainData._formFactory = new ClassFormFactory();
            _mainData.IMouseHandler = new DrawFromMouseHandler();
        }
        if (e.Control && e.KeyCode == Keys.I)
        {
            _mainData._formFactory = new InterfaceFormFactory();
            _mainData.IMouseHandler = new DrawFromMouseHandler();
        }
        if (e.Control && e.KeyCode == Keys.N)
        {
            if (_mainData.FormsList.Count != 0)
            {
                const string infoText = "Создание нового файла удалит несохраненные изменения.
Сохранить изменения?";
                const string warningText = "Внимание!";
                DialogResult dialogResult = MessageBox.Show(infoText, warningText,
MessageBoxButtons.YesNoCancel, MessageBoxIcon.Warning);
                if (dialogResult == DialogResult.Yes)
                {
                    toolStripButtonSaveFile_Click(sender, e);
                }
                else if (dialogResult == DialogResult.No)
                {
                    _mainData.FormsList = new List<AbstractForm>();
                    _mainData.ArrowsList = new List<Arrow>();
                    _mainData.PictureBoxMain.Invalidate();
                }
            }
        }
        if (e.Control && e.KeyCode == Keys.A)
        {
            statusStripStatusLabel.Text = "Click to start point and drag mouse to end point, to crate arrow
association";
            _mainData._arrowsFactory = new ArrowAssociationFactory();
            _mainData.IMouseHandler = new DrawArrowMouseHandler();
        }
        if (e.Control && e.KeyCode == Keys.M)
        {
            _mainData._arrowsFactory = new ArrowCompositionFactory();
            _mainData.IMouseHandler = new DrawArrowMouseHandler();
        }
        if (e.Control && e.KeyCode == Keys.R)
        {
            _mainData._arrowsFactory = new ArrowRealizationFactory();
            _mainData.IMouseHandler = new DrawArrowMouseHandler();
```

```
    }

    if (e.Control && e.KeyCode == Keys.S)
    {
        _mainData._arrowsFactory = new ArrowSuccessionFactory();
        _mainData.IMouseHandler = new DrawArrowMouseHandler();
    }
    if (e.Control && e.KeyCode == Keys.X)
    {
        this.Close();
    }

}

private void saveFileToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (saveFileDialog1.ShowDialog() == DialogResult.OK)
    {
        string fileDataForms = JsonSerialize(TypeOfData.Forms);
        string fileDataArrows = JsonSerialize(TypeOfData.Arrows);
        SaveAndLoad.SaveFile(saveFileDialog1.FileName, fileDataForms, fileDataArrows);
    }
}

private void closeProgramToolStripMenuItem_Click(object sender, EventArgs e)
{
    this.Close();
}

private void copyObjectsToolStripMenuItem_Click(object sender, EventArgs e)
{
    copyToStackButton_Click(sender, e);
}

private void pasteObjectsToolStripMenuItem_Click(object sender, EventArgs e)
{
    pictureBoxMain.MouseDown += PasteObject_MouseDown;
}

private void editObjectsToolStripMenuItem_Click(object sender, EventArgs e)
{
    FormEditor formEditor = new FormEditor();
    formEditor.Show();
    _mainData.IMouseHandler = new SelectFormMouseHandler();
}

private void aggregationToolStripMenuItem_Click(object sender, EventArgs e)
{
    _mainData._arrowsFactory = new ArrowAggregationFactory();
    _mainData.IMouseHandler = new DrawArrowMouseHandler();
}

private void realizationToolStripMenuItem_Click(object sender, EventArgs e)
{
    _mainData._arrowsFactory = new ArrowRealizationFactory();
    _mainData.IMouseHandler = new DrawArrowMouseHandler();
}

private void successionToolStripMenuItem_Click(object sender, EventArgs e)
{
    _mainData._arrowsFactory = new ArrowSuccessionFactory();
    _mainData.IMouseHandler = new DrawArrowMouseHandler();
}
```

```
        private void shotkeyListToolStripMenuItem_Click(object sender, EventArgs e)
        {
            MessageBox.Show("Shot key in UML bilder\n" +
                "Ctrl+C - Copy object\n" +
                "Ctrl+P - Paste object\n" +
                "Ctrl+O - Open UML-diagram\n" +
                "Ctrl+S - Save UML-diagram\n" +
                "Ctrl+J - Save UML-diagram in JPEG\n" +
                "Ctrl+N - Create new UML-diagram\n" +
                "Ctrl+X - Close UML-diagram\n" +
                "Ctrl+I - Add class\n" +
                "Ctrl+A - Add aggregation arrow\n" +
                "Ctrl+I - Add interfase arrow\n" +
                "Ctrl+M - Add composition arrow\n" +
                "Ctrl+R - Add Realization arrow\n" +
                "Ctrl+S - Add succession arrow\n"
                );

        }

        private void propertiesToolStripMenuItem_Click(object sender, EventArgs e)
        {
            MessageBox.Show("How to create UML-diagram?\n" +
                "If you want create UML-diagram, you should set up on the desk all classes and interfases\n" +
                "and create dependances for all classes and interfases\n" +
                "How to save UML-diagram?\n" +
                "If you want create UML-diagram, you should mouse down on button Save to Jpeg or Save to\n" +
                "or Keydown short key Ctrl+S OR Ctrl+J\n");
        }
    }
}
```

ДОДАТОК Б

ВІДГУК

керівника економічного розділу
на кваліфікаційну роботу бакалавра
на тему:
«Розробка графічного редактору «UML-Diagram drawer» для створення та редагування UML-діаграм з використанням .NET»
студента групи 122-19ск-2 Соколова Артура Вікторовича

Керівник економічного розділу
доцент каф. ПЕП та ПУ, к.е.н **Л. В. Касьяненко**

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файла	Опис
Пояснювальні документи	
Диплом.doc	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Диплом.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF.
Програма	
Diplom.zip	Архів. Містить коди програми і откомпільовану програму.
Презентація	
Презентація.ppt	Презентація кваліфікаційної роботи.