Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет «Комп'ютерні технології і системи»

Кафедра «Комп'ютерні інформаційні технології»

Пояснювальна записка
до кваліфікаційної роботи бакалавра

на тему: «Розробка CASE-додатку з підтримкою проектного підходу»
за освітньою програмою: «Інженерія програмного забезпечення»
зі спеціальності: «121 Інженерія програмного забезпечення»
Виконав: студент групи «ПЗ1811»

_____ /Владислав БАЖИН/
(підпис студента) (Ім'я ПРІЗВИЩЕ)

Керівник: _____ /доц. Олена КУРОП'ЯТНИК/
(підпис) (посада, Ім'я ПРІЗВИЩЕ)

Нормоконтролер: _____ /доц. Олена КУРОП'ЯТНИК/
(підпис) (посада, Ім'я ПРІЗВИЩЕ)

Засвідчую, що у цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Дніпро – 2022 рік

Ministry of Education and Science of Ukraine
Ukrainian State University of Science and Technologies

Faculty «Computer technologies and systems»

Department «Computer information technology»

Explanatory Note
to Bachelor's Thesis

on the topic: «Development of CASE-application with project approach support»
according to educational curriculum « Software engineering »
in the Speciality: «121 Software engineering»

Done by the student of the group PZ1811: /Vladyslav BAZHYN/

Scientific Supervisor: /Olena KUROPIATNYK/

Normative controller: /Olena KUROPIATNYK/

Dnipro – 2022

Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет: Факультет «Комп'ютерні технології і системи»
Кафедра: «Комп'ютерні інформаційні технології»
Рівень вищої освіти: бакалавр
Освітня програма: «Інженерія програмного забезпечення»
Спеціальність: «121 Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри КІТ
_____ /Вадим ГОРЯЧКІН/
(підпис)

Дата _____

ЗАВДАННЯ

на кваліфікаційну роботу бакалавра
студенту Бажину Владиславу Андрійовичу
1. Тема роботи: «Розробка CASE-додатку з підтримкою проектного підходу»
Керівник роботи: Куроп'ятник Олена Сергіївна, доцент
затверджені наказом № 77 ст від 08.12.2021
2. Строк подання студентом роботи: __.__.202_ р.
3. Вихідні дані до роботи: _____

4. Зміст пояснювальної записки (перелік питань, які потрібно опрацювати):
Вступ
Збір вимог до програмного забезпечення
Зовнішнє і внутрішнє проектування
Тестування та налагодження
Висновки
Список літератури
Додатки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень):
Презентація, що містить 15 слайдів

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Постановка задачі, збір вимог до програмного забезпечення	31.01.22	
2	Огляд основних аналогів	10.02.22-12.02.22	
3	Розробка програми	01.03.22-31.03.22	
4	Тестування та налагодження програми	01.04.22-10.04.22	
5	Написання основного змісту пояснювальної записки	15.04.22-01.05.22	
6	Узгодження і затвердження програмної документації	31.05.22-12.06.22	
7	Подання кваліфікаційної роботи до кафедри	07.06.22	
8	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії	21.06.22	

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

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

РЕФЕРАТ

Пояснювальна записка складається з 7 розділів:

– вступ – в даному розділі описується сутність розробки, її актуальність. Складається з 1 сторінки;

– збір вимог до програмного забезпечення – у цьому розділи описуються аналоги програми та література по даній предметній області, а також проводиться опитування зацікавлених сторін для формування найбільш повних вимог до програмного забезпечення. Складається з 13 сторінок;

– зовнішнє і внутрішнє проектування – у цьому розділі проведений огляд вхідних і вихідних даних, формалізація задачі, розробка фізичного проекту, приводиться опис об'єктно-орієнтованого проектування, проектування інтерфейсу користувача, ескізи форм, аналіз проекту, проектування динаміки системи, вибір мови програмування. Складається з 12 сторінок;

– тестування та налагодження – включає в себе вибір стратегії тестування, опис тестів методами «чорного» та «білого» ящика. Також аналіз помилок їх вплив на систему та вирішення проблеми. Складається з 10 сторінок;

– висновки. Складається з 1 сторінки;

– список літератури – включає в себе бібліографічний список використаної літератури. Складається з 1 сторінки;

– додатки – містить код програми.

Кількість таблиць: 6 штук. Кількість рисунків: 16 штук.

Ключові слова: CASE, елементи, діаграми, UML, ООП, Visual Studio, C#, система.

ЗМІСТ

ВСТУП

Суть розробки проекту полягає у створенні CASE-додатку з підтримкою проектного підходу. Поняття «CASE» передбачає у собі сукупність методів та інструментів програмної інженерії для створення різних проектів й розробки програмного забезпечення. Ці методи допомагають покращити якість програм, уникати помилок й гарантують простоту у використанні.

За допомогою програмного забезпечення, що розроблюється, можна буде створювати різноманітні діаграми. Іншими словами, це – UML-редактор. У даній програмі користувач матиме можливість створювати, редагувати, зберігати свої діаграми. Також буде застосований проектний підхід. Він дозволяє користувачу використовувати елементи однієї своєї діаграми у іншій, тобто значно пришвидшувати та покращувати свою роботу.

Актуальність роботи: на сьогоднішній день дана робота є досить актуальною, оскільки діаграми використовуються при розробці багатьох проектів. Це допомагає вже на етапі проектування розуміти, що потрібно і як зробити, тому велика кількість розробників та компаній зараз використовує різні редактори для побудови діаграм. Проте, окрім професійних розробників UML-редакторами також користуються викладачі та студенти, а майже всі аналоги ринку платні й мають досить обмежений безкоштовний функціонал. Тому, розробка даного проекту допоможе людям зі сфери освіти створювати діаграми без обмеженого функціоналу безкоштовно. Також під час розробки даного проекту, було розглянуто переваги та недоліки існуючих аналогів та проведено опитування зацікавлених сторін, завдяки чому було покращено кінцевий продукт.

РОЗДІЛ 1 ЗБІР ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Огляд аналогів

В ході розробки програми дипломної роботи, було розглянуто аналоги UML-редакторів, визначено їх переваги та недоліки, покращення власної програми завдяки висновкам, зроблених під час огляду аналогів. Одними з найпопулярніших редакторів є веб-сайти. Частина функціоналу безкоштовна. Проте, для користування повним функціоналом потрібно оформити платну підписку. До уваги обрано найпопулярніші продукти ринку. Далі подано результати їх огляду.

Також серед аналогів можна відзначити спеціальні програми для побудови UML-діаграм. Тим не менш, більшість таких програм вже досить застарілі й не підтримуються на більш нових версіях операційних системи. Гарним прикладом виступає один з розглянутих аналогів – Rational Rose.

Веб-сайт Creately.com Загальний функціонал та інтерфейс: сайт складається з реєстраційної форми з можливістю одразу увійти під своїм google-акаунтом без реєстрації. Після входу у свій акаунт, користувач обирає наступний етап: працювати зі старими документами, або створити новий. Із основного: завдяки цьому продукту користувач має можливість побудувати будь-яку UML-діаграму, блок-схеми, а також додаткові проектні засоби для створення шаблонів проектів й планування роботи. Ще надається можливість обрати вже готовий шаблон для подальшого створення діаграми, що прискорює процес побудови деяких варіантів. Є багато можливостей для роботи: велика кількість об’єктів для розробки будь-яких діаграм, додаткові стікери, блоки та фігури тощо. У фунціоналі є можливість зберігати свої проекти й працювати з чернеткою вже існуючих макетів.

Переваги: під час використання користувачу надається можливість переглянути відео-урок з роботою цього продукту. Досить значна частина контенту – безкоштовна. Зручна система збереження документів. Дуже великий функціонал, завдяки чому користувач має багато можливостей. Є

зворотній зв'язок з розробником та технічна підтримка, що може допомогти з проблемами під час використання сайту.

Недоліки: важко розібратися з усіма можливостями без перегляду відео-уроків. Мала кількість файлів, які можна зберігати у безкоштовній версії, неможливість переглядати свою історію, відсутність функціоналу, пов’язаного з використанням бази даних тощо. Даний продукт більше підходить для великих компаній, що планують використовувати даний продукт довгий час та мають багато працівників. Менше ця програма слугує для постійного використання студентами, чи викладачами, оскільки для зручного постійного використання буде дуже важко працювати без платної підписки з повним функціоналом.

Головну сторінку веб-сайту Creately.com з її елементами можна побачити на рисунку 1.1:

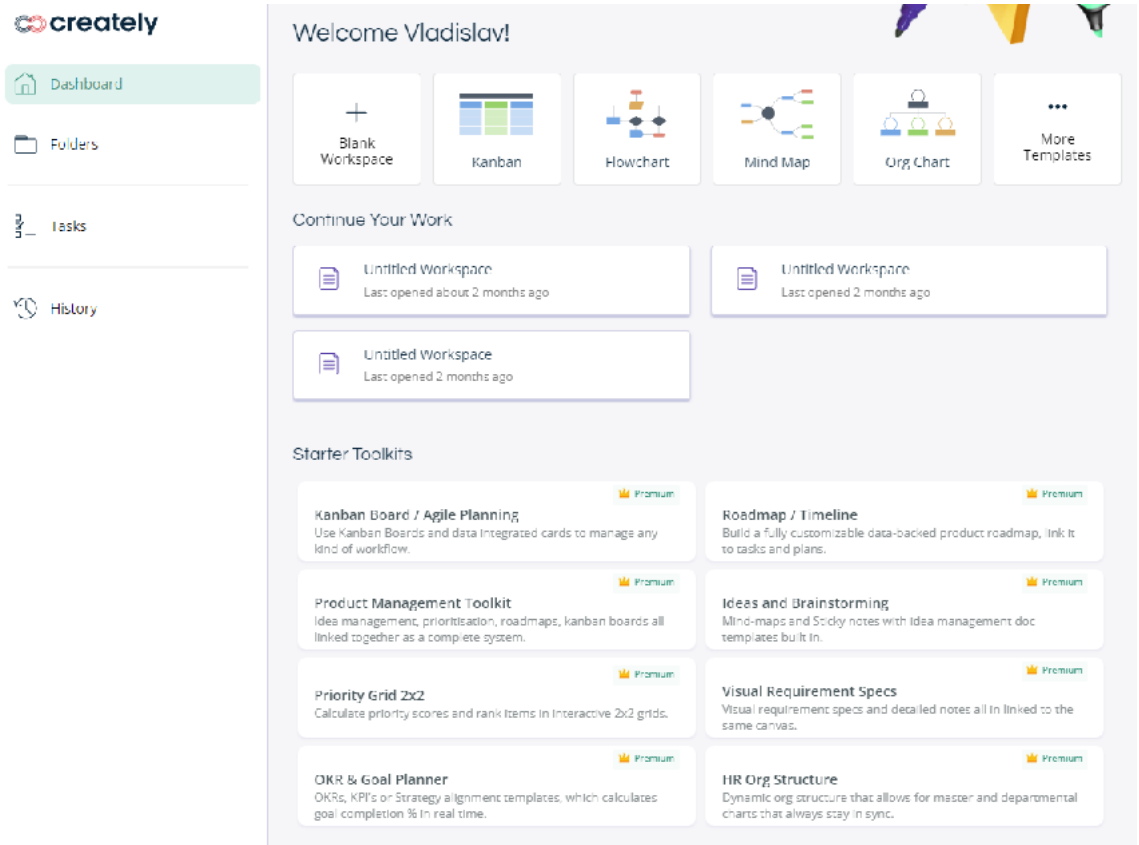

Рисунок 1.1 – Головна сторінка веб-сайту

Основний функціонал веб-сайту Creately.com з усіма можливостями та інструментами для побудови UML-діаграм можна побачити на рисунку 1.2:

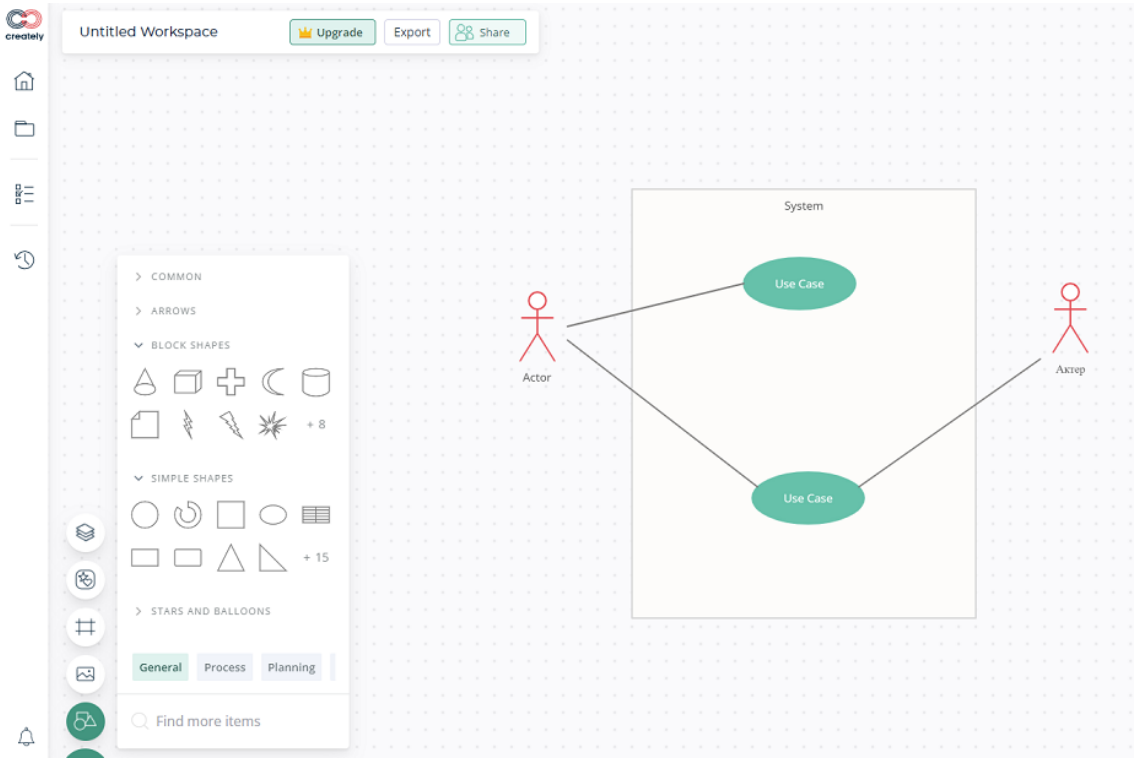

Рисунок 1.2 – Форма для побудови UML-діаграм

Веб-сайт Lucid.app Загальний функціонал та інтерфейс: реєстрація здійснюється так само, як і в попередньому аналогу. Більш простий інтерфейс, що має певні відмінності, проте суть не змінюється, лише деякі різності у можливостях цих двох продуктів, віддаючи перевагу першому. Проте, присутні безкоштовні функції, що недоступні у першому варіанті. Наприклад, перегляд своєї історії та збереження більшої кількості файлів. Основним критерієм являється можливість побудови різноманітних діаграм, що також є у даному продукті. Немає «перевантаженості» інтерфейсу, як у попередньому аналогу. Присутня можливість збереження готових документів,

Переваги: зручний та простий інтерфейс, в якому легко розібратися. Більшість функціоналу зрозуміла на інтуїтивному рівні. Багато можливостей доступні безкоштовно. Присутні підказки, що допомагають розібратися під час використання сайту. Присутня інтеграція з іншими популярними сервісами, такими як: slack, Microsoft Teams та Google Drive. Зручна форма для побудови UML-діаграм, що значно пришвидшує процес побудови й покращує якість остаточних результатів.

Недоліки: дещо менший функціонал, порівняно з попереднім аналогом. Менша пам'ять для збереження файлів, кількість об’єктів на документ, деякі інші функції доступні лише у платній версії. Відсутність проектного підходу, що значно полегшує роботу. Неможливість будувати деякі види UML-діаграм, що є великим мінусом. Даний веб-сайт підтримує можливість побудови UML-діаграм, проте це не є його основною метою. Окрім інструментів для побудови, веб-сайт також містить інші можливості для проектування тощо.

Головну сторінку веб-сайту Lucid.app з її елементами можна побачити на рисунку 1.3:

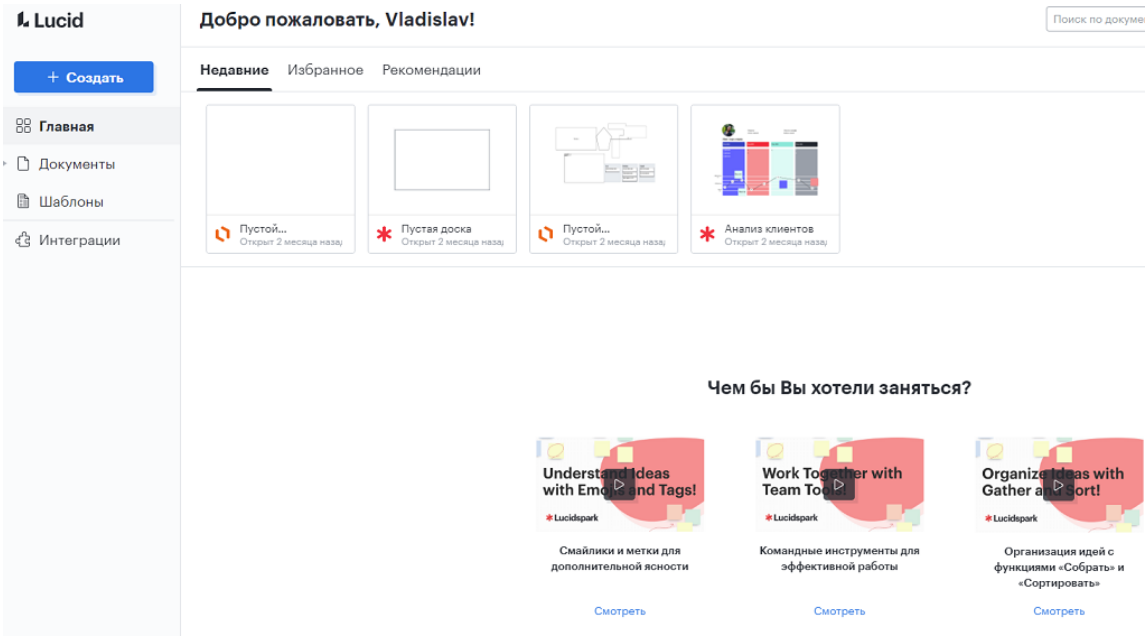

Рисунок 1.3 – Головна сторінка веб-сайту

Основний функціонал веб-сайту Lucid.app з усіма можливостями та інструментами для побудови UML-діаграм можна побачити на рисунку 1.2:

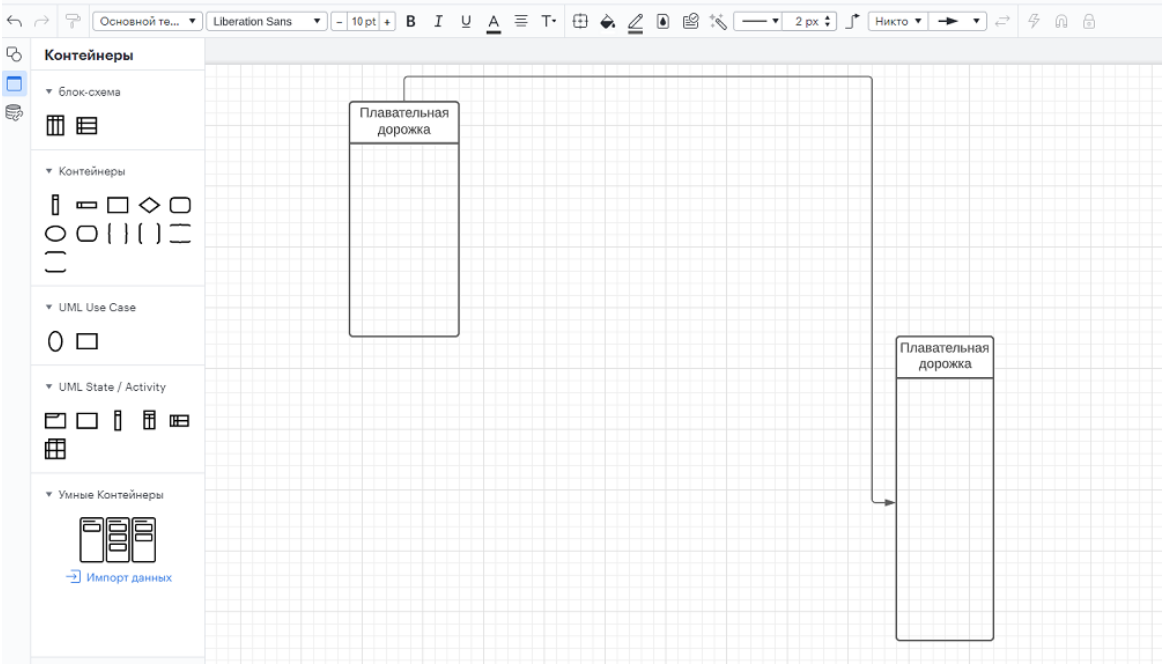

Рисунок 1.4 – Форма для побудови UML-діаграм

Програма Rational Rose Програмне забезпечення, що не потребує реєстрації для її використання. Досить простий інтерфейс, в якому усі головні CASE-елементи програми знаходяться на формі. За допомогою програми є можливість побудувати будь-які UML-діаграми. Можна зберегти робочий файл для подальшого використання, або як вже готову UML-діаграму. Являється одною з найперших програм для побудови UML-діаграм з підтримкою проектного підходу.

Переваги: підтримка проектного підходу, що дозволяє використовувати елементи зі вже готової UML-діаграми при побудові іншої. Можливість збереження та продовження роботи. Досить великий функціонал, що дозволяє використовувати програми як для освіти, так і для компаній, що зацікавлені у проектуванні та розробці програмного забезпечення. Є можливість переглянути файл з інструкцією по роботі з даним програмним забезпеченням.

Недоліки: досить застарілий інтерфейс у порівнянні з більш новими аналогами. Не зовсім зручне «скупчення» усіх елементів на одній формі, що створює труднощі при використанні програми без досвіду. Застарілі функції

роботи з UML-діаграмами. Відсутність інтеграції, або більш нових версій для роботи з новими версіями операційних систем, що робить дану програму неможливою для використання у теперішній час.

Основний функціонал програми Rational Rose з усіма можливостями та інструментами для побудови UML-діаграм можна побачити на рисунку 1.5:

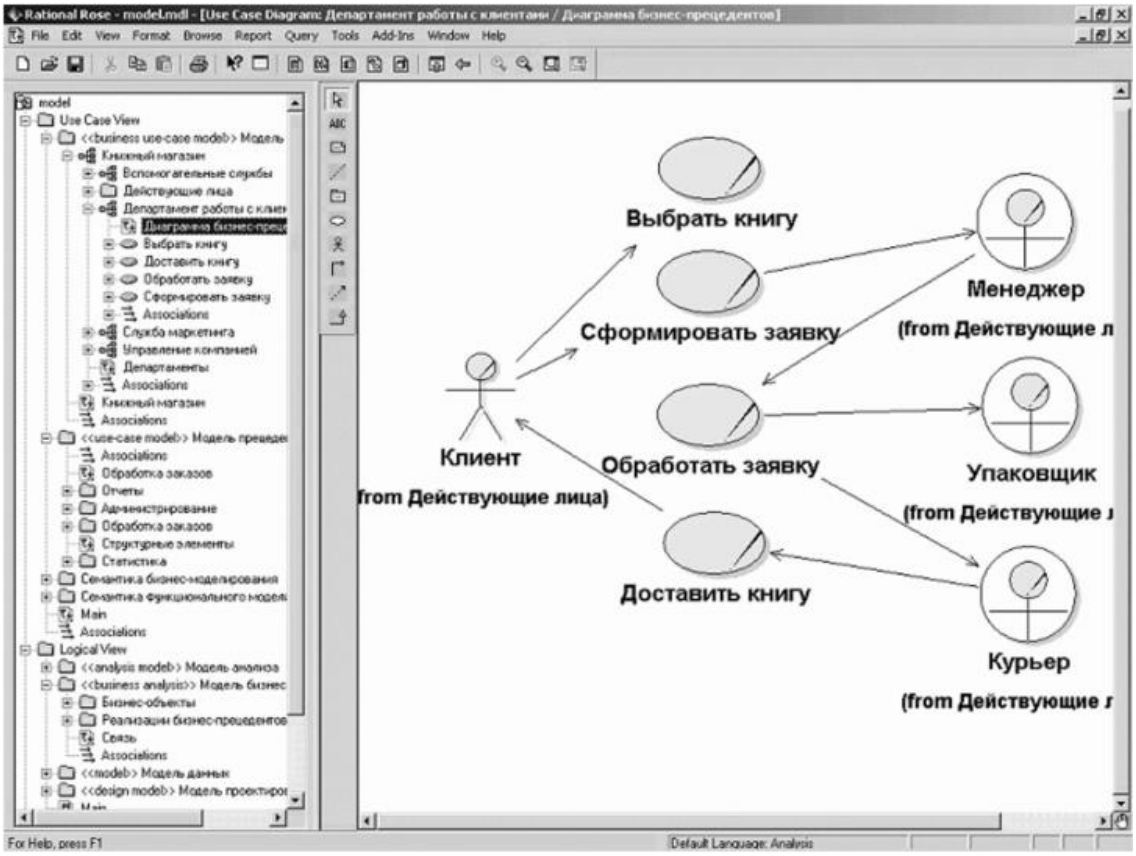

Рисунок 1.5 – Основна форма програми з її елементами

1.2 Опитування зацікавлених сторін

Опитування проводилося за допомогою google-форми. Було представлено основні питання, які допомогли би під час подальшого проектування програмного забезпечення. Усього було опитано 28 респондентів. До основних питань було віднесено досвід використання, додатковий функціонал, сферу діяльності респондентів та які б основні види діаграм вони хотіли бачити у даному проекті. Детальні результати представлено на рисунках 1.6 - 1.9:

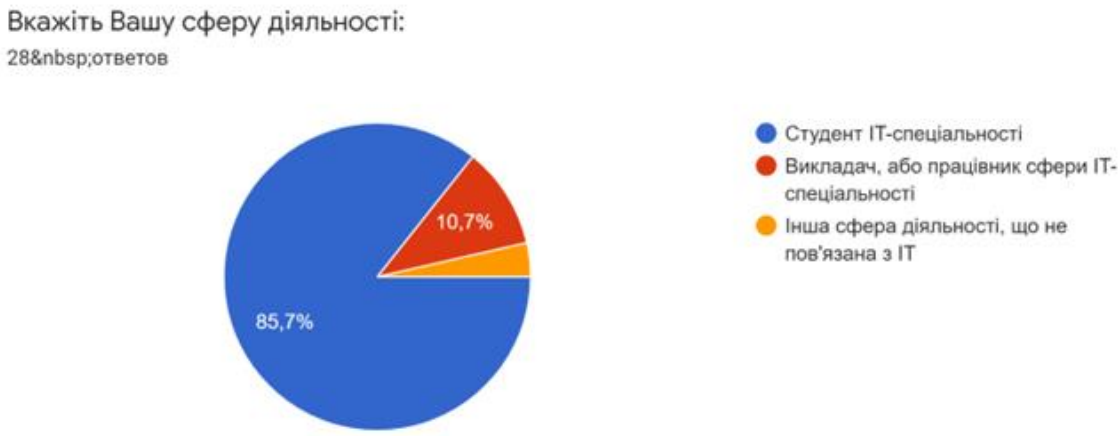

Рисунок 1.6 – Сфера діяльності

Більшість опитаних респондентів являються студентами IT-спеціальності, тому після повної розробки продукту та при можливості подальшого використання у більш широкому форматі, було б доцільніше зробити безкоштовний функціонал для студентів.

Рисунок 1.7 – Досвід використання

У більшості респондентів вже є досвід з використання аналогів. Серед людей без досвіду були студенти молодших курсів, а також люди, які не пов’язані зі сферою IT. Тому, буде доцільним додати до програмного

забезпечення посібник користувача, щоб кожен працюючий з програмою міг з легкістю розібратися в основному функціоналі.

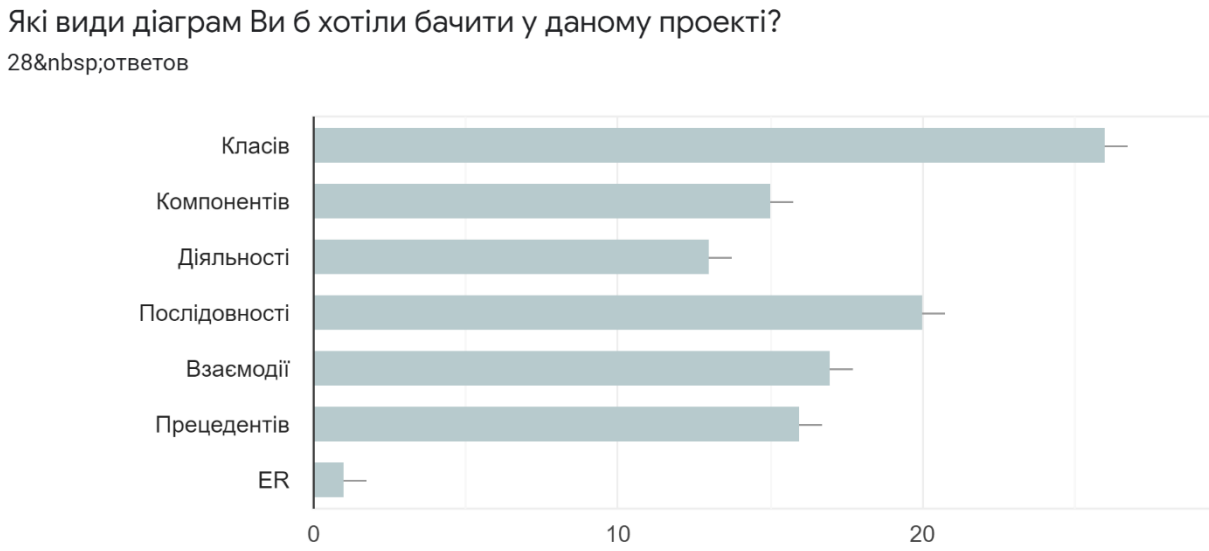

Рисунок 1.8 – Види діаграм

Основною потребою в UML-редакторі є діаграма класів, тому перш за все буде розроблятися саме можливість підтримки діаграми класів. Менш за все респонденти зацікавлені у діаграмах компонентів та діяльності. За ER-діаграму не було голосів.

Дане питання добре показує на що потрібно поставити пріоритети при початковій розробці та тестуванні: перш за все, можливість побудови діаграми класів. Після додавання інструментів для побудови перших видів UML-діаграм до програмного продукту можна буде приступати до розробки й реалізації наступних видів діаграм.

Основна кількість опитаних респондентів бажають бачити у програмі наявність шаблонів діаграм, підтримку проектного підходу, а також можливість налаштування кольорової схеми / шрифту. Це саме той функціонал, який допомагає будь-якому користувачу з легкістю користуватися програмою без особливих складнощів.

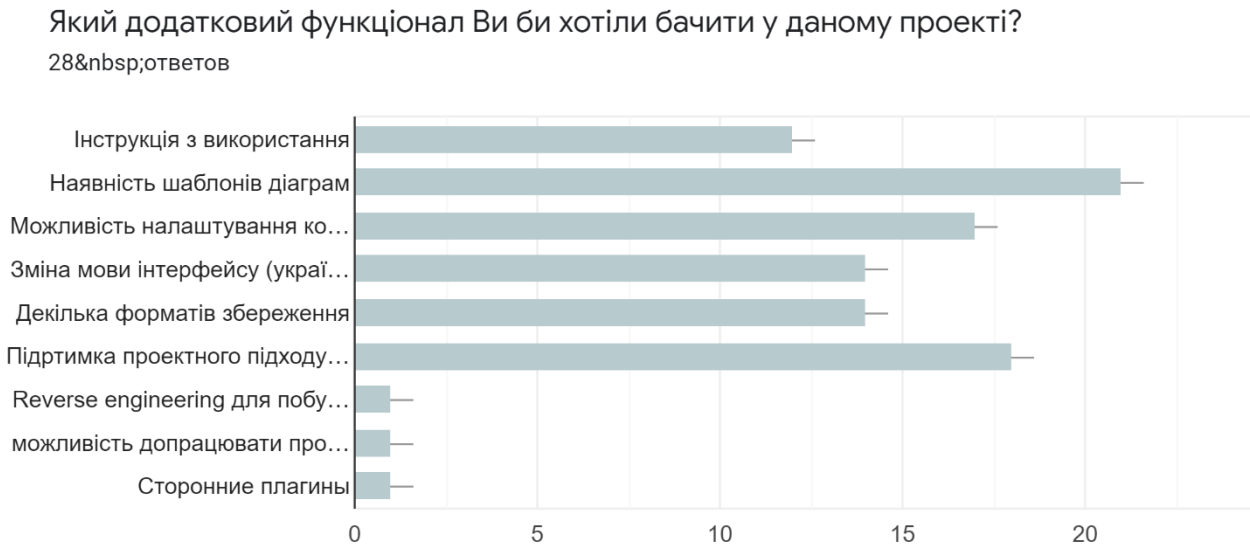

Рисунок 1.9 – Додатковий функціонал

Також не менше голосів респонденти віддали на інструкцію з використання, зміни мови інтерфейсу програми й декілька форматів збереження остаточного файлу. Тому, саме цьому функціоналу буде надано перевагу після розробки основних елементів UML-редактору.

1.3 Огляд літератури

Існує декілька підходів до розробки програмного забезпечення з використанням CASE-засобів, які в свою чергу також мають декілька підходів, один з яких об’єктно-орієнтований. У більш сучасних компаніях технології розробки намагаються усюди перевести саме на об’єктно-орієнтований підхід, тому є декілька причин:

– можливістю складання програмної системи з готових компонентів, які можна використати повторно;

– можливістю накопичення проектних рішень у вигляді бібліотек класів на основі механізмів успадкування;

– простотою внесення змін до проектів за рахунок інкапсуляції даних в об'єктах;

– швидкої адаптацією додатків до змінних умов за рахунок використання властивостей успадкування та поліморфізму;

– можливістю організації паралельної роботи аналітиків, проектувальників та програмістів.

«Концепції об'єктно-орієнтованого підходу та розподілених обчислень стали базою для створення консорціуму Object Management Group (OMG), членами якої є понад 500 провідних комп'ютерних компаній (Sun, DEC, IBM, HP, Motorola та ін.). Основним напрямом діяльності консорціуму є розробка специфікацій та стандартів для створення розподілених об'єктних систем у різнорідних середовищах» [2]. Базисом стали специфікації за назвою Object Management Architecture (OMA).

OMA складається з чотирьох основних компонентів, що представляють специфікації різних рівнів підтримки програм:

– архітектура брокера запитів об'єктів (CORBA – Common Object Request Broker Architecture) визначає механізми взаємодії об'єктів у різнорідній мережі;

– об'єктні послуги (Object Services) є основними системними сервісами, використовуваними розробниками для створення додатків;

– універсальні засоби (Common Facilities) є високорівневими системними сервісами, орієнтованими на підтримку додатків користувача (електронна пошта, засоби друку та ін.);

– прикладні об'єкти (Application Object) призначені для вирішення конкретних прикладних завдань.

Виходячи з основних положень об'єктно-орієнтованого підходу, розглянемо концепцію ідеального об'єктно-орієнтованого CASE-засобу [1].

Існує кілька об'єктно-орієнтованих методів, авторами найбільш поширених є Г. Буч, Д. Рамбо, І. Джекобсон. Нині спостерігається процес зближення об'єктно-орієнтованих методів. Зокрема, зазначені вище автори створили та випустили кілька версій уніфікованого методу UML (Unified Modeling Language – уніфікована мова моделювання).

Класична постановка задачі розробки програмної системи (інжиніринг) є спіральним циклом ітеративного чергування етапів об'єктно-орієнтованого аналізу, проектування та реалізації (програмування).

У реальній практиці здебільшого є передісторія у вигляді сукупності розроблених та впроваджених програм, які доцільно використовувати при розробці нової системи. Процес проектування у разі заснований на реінжинірингу програмних кодів, у якому шляхом аналізу текстів програм відновлюється вихідна модель програмної системи.

Сучасні CASE-засоби підтримують процеси інжинірингу та автоматизованого реінжинірингу.

Ідеальний об'єктно-орієнтований CASE-засіб повинен містити чотири основні блоки: аналіз, проектування, розробка та інфраструктура.

Порівняльний аналіз CASE-систем показує, що на сьогоднішній день одним із найбільш наближених до ідеального варіанту CASE-засобів є сімейство Rational Rose фірми Rational Software Corporation. Слід зазначити, що тут працюють автори уніфікованої мови моделювання Г. Буч, Д. Рамбо та І. Джекобсон, під керівництвом яких ведеться розробка нового CASE-засобу, що підтримує UML.

Виділимо основні критерії оцінки та вибору CASE-засобів.

1) функціональні характеристики:

a) середовище функціонування: проектне середовище, програмне забезпечення/технічні засоби, технологічне середовище;

b) функції, орієнтовані фази життєвого циклу: моделювання, реалізація, тестування;

c) загальні функції: документування, управління конфігурацією, управління проектом;

2) надійність;

3) простота використання;

4) ефективність;

5) супроводжуваність;

6) переносимість.

1.4 Постановка задачі

Одною з найперших задач являється огляд аналогів, визначення загальних переваг та недоліків основних UML-редакторів ринку, зробити висновки, як можна покращити власний проект на основі отриманої інформації.

Розробити власний додаток, опираючись на основні потреби користувачів, що зацікавлені у даному проекті й покращити існуючий функціонал. Провівши опитування зацікавлених сторін, підкреслити для себе найголовніші потреби користувачів та додати до основного функціоналу.

Додати до існуючого продукту проектний підхід – метод, що допомагає користувачу легше працювати з програмою. Під час його використання, користувач має змогу використовувати елементи однієї UML-діаграми, побудованої раніше, у вже новій, щоб полегшити процес роботи.

Основні задачі:

– можливість побудови UML-діаграм;

– завантаження готової UML-діаграми;

– збереження розробленої UML-діаграм.

Додаткові задачі:

– зробити можливість генерації коду на основі побудованих UML-діаграм;

– додати шаблони готових проектів;

– додати підтримку зміни мови та шрифту;

– зробити інструкцію з використання.

Висновки до розділу 1

Збір вимог під час проектування програмного забезпечення є одною з найперших та найосновніших задач. Це допомагає якісно оцінити вимоги та спланувати подальшу роботу. Підкреслює основні недоліки аналогів ринку та додає більше переваг до кінцевого продукту.

Під час збору вимог було опитано зацікавлені сторони (викладачів, студентів та працівників й людей зі сфери IT-технологій). Складено статистику основних потреб користувачів, а також зібрано інформацію про досвід використання схожих програм. Опираючись на отриману інформацію, створено план подальшої роботи.

РОЗДІЛ 2 ЗОВНІШНЄ І ВНУТРІШНЄ ПРОЕКТУВАННЯ

2.1 Визначення вхідних і вихідних даних

Програма повинна забезпечити можливість побудови діаграм з підтримкою проектного підходу.

До неї належать такі основні сутності: CASE-елементи;

В програмі необхідно реалізувати збереження у файлі.

Вхідні данні: елементи UML-діаграми: основний діючий елемент (наприклад, клас-1 у діаграмі класів, користувач у діаграмі прецедентів тощо), зв'язки у різних видах діаграм.

Діаграма будується за допомогою елементів форми.

Вихідні дані: побудована UML-діаграма у вигляді jpg-файлу.

Самі файли зберігаються у документі (окрема папка).

Вихідні дані виводяться у вигляді інтерфейсу.

У програмі повинен бути реалізований проектний підхід – коли під час побудови однієї діаграми є можливість використовувати дані, що були використані під час побудови попередньої діаграми.

Вимоги до надійності наступні:

– повідомлення про те, що деякі поля були не заповнені;

– попередження, якщо користувач закриває програму без збереження й додатково пропонує зберегти поточний файл;

– копія коду програми на додатковому зовнішньому носії.

Схема варіантів використання показана на рисунку 2.1:

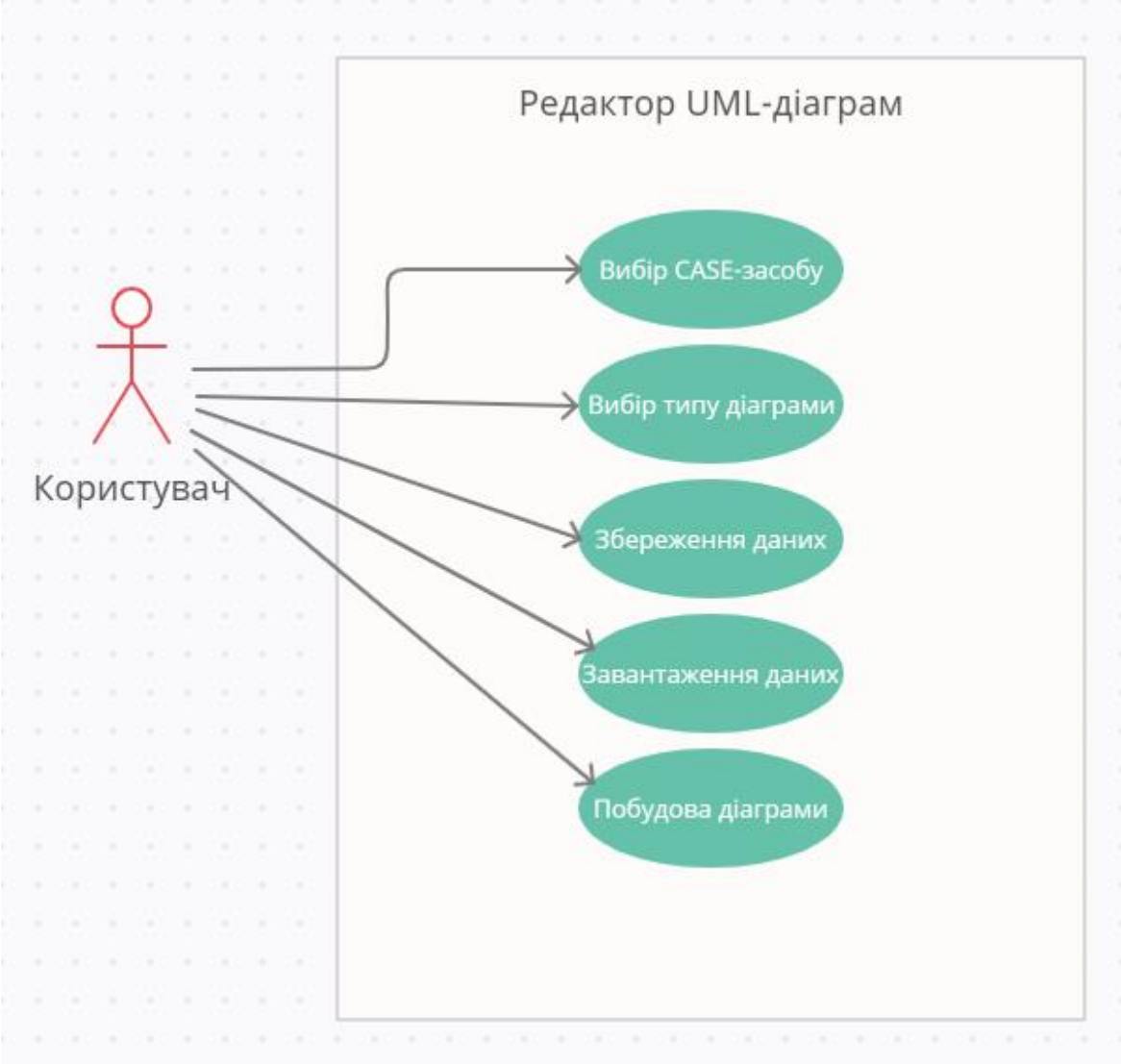

Рисунок 2.1 – Діаграма прецедентів

На даній діаграмі прецедентів показано, що у користувача є вибір:

– війти у систему (запуск програми);

– вибір CASE-засобу (інструментів для створення UML-діаграм);

– вибір типу діаграми (класів, послідовності тощо);

– збереження даних (збереження UML-діаграми в окремому файлі;

– вихід з системи (вихід із програми).

2.2 Призначення розробки

Функціональне призначення – програмний продукт являє собою програму для побудови різноманітних UML-діаграм.

Діаграм буде 6 видів:

– класів;

– компонентів;

– діяльності;

– послідовності;

– взаємодії;

– прецедентів.

Експлуатаційне призначення – побудова діаграм з підтримкою проектного підходу, коли можна використовувати дані з однієї діаграми при побудові іншої, а також збереження у файлі.

2.3 Розробка фізичного проекту

Давайте детальніше розглянемо основні процеси, які здійснюються в системі. Для цього використовують діаграми стану. На рисунку 2.2 показана діаграма станів проектованої системи.

Після того, як користувач заходить до програми, він обирає яку UML-діаграму він буде будувати, після цього обирає елементи цієї діаграми, розставляючи їх на формі. Зробивши усе необхідне, або ври спробі вийти з програми буде запропоновано збереження файлу. Під кінець роботи користувач має можливість переглянути збережений файл, або відкрити його й почати роботу з ним.

2.4 Проектування інтерфейсу користувача

Інтерфейс користувача розроблений за допомогою Windows Form. Він складається лише с 2 класів: Form1, Program. На формі знаходяться такі елементи керування, як button, richtextbox, combobox, label та textbox. Опис методів даного модулю представлений в таблиці 2.1.

Основні використані елементи керування інтерфейсом є:

– Form – головний елемент інтерфейсу програмного забезпечення, на якому розташовуються всі інші елементи;

– button – елемент інтерфейсу кнопка, який при натисканні починає виконувати функції, що написані в програмі;

– label – елемент інтерфейсу, який лише виводить інформацію, немає можливість редагувати на формі;

– textbox – елемент інтерфейсу, що записує, редагує та виводить інформацію в залежності з роботи програми;

– richtextbox - елемент інтерфейсу, що записує, редагує та виводить інформацію в залежності з роботи програми, відрізняється в textbox заданим розміром елементу;

– combobox – елемент інтерфейсу, який дає можливість перегляду списку та вибирати потрібний елемент [3].

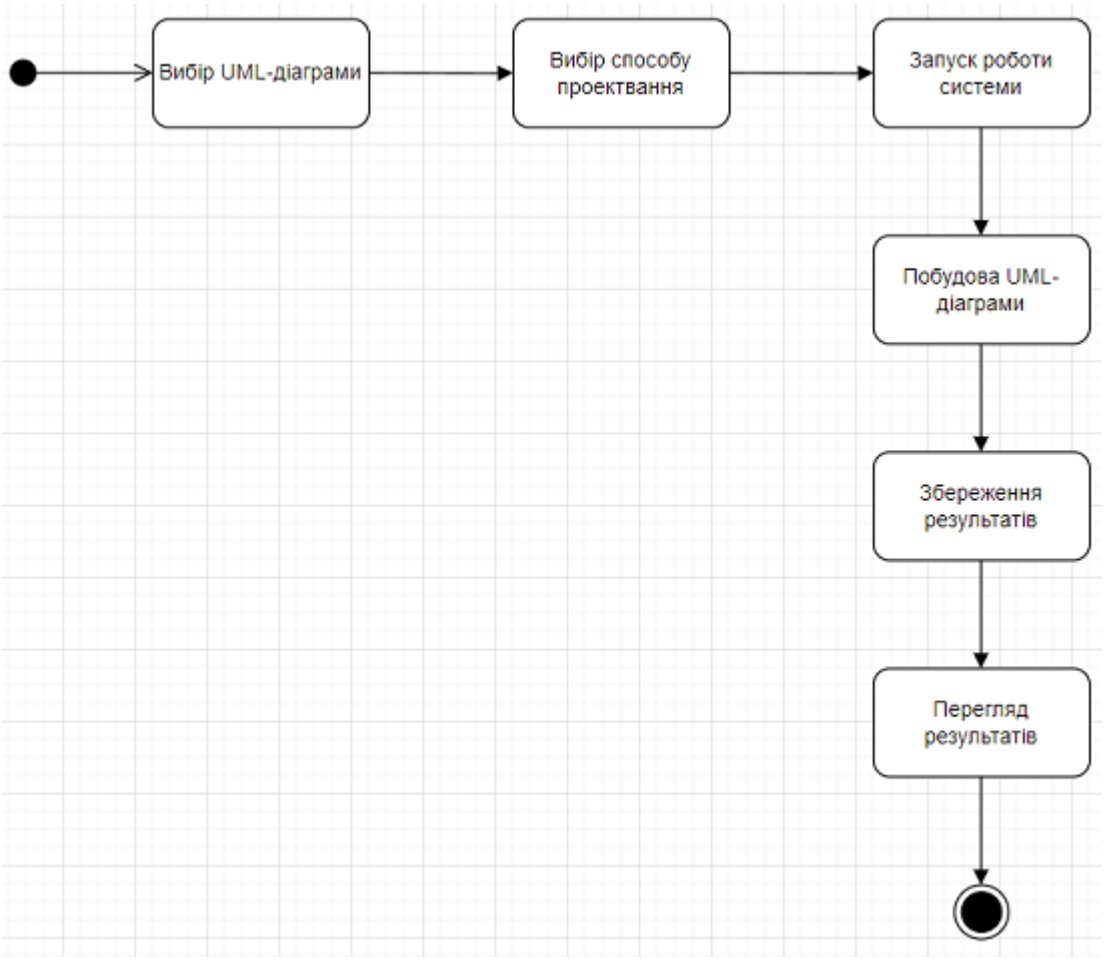

Рисунок 2.2 – Діаграма станів системи

Таблиця 2.1 – Опис класів та методів модулю інтерфейсу

Назва класу	Метод	Параметри Тип значення, що повертається	Опис
1	2	3	4
Form1	button1_Click()	Object sender, EventArgs e void	Реагує на натиск кнопки з відповідним кейс-елементом і виводить результат у відповідне поле
	button2_Click()	Object sender, EventArgs e void	Реагує на натиск кнопки «Nodes» та виконує і виводить результат у відповідне поле

Продовження таблиці 2.1

1	2	3	4
	button4_Click()	Object sender, EventArgs e void	Реагує на натиск кнопки «Завантажити файл з системи» другого файлу та заносить назву файлу у відповідне поле
	button3_Click_1()	Object sender, EventArgs e void	Реагує на натиск кнопки «Завантажити файл з системи» першого файлу та заносить назву файлу у відповідне поле
	button7_Click()	Object sender, EventArgs e void	Реагує на натиск кнопки «Зберегти результати». Зберігає результати у окремому файлі
Program	Main()	void	Запускає клас Form1

2.5 Ескізи форм

Початковий ескіз, створений для форми програми можна побачити на рисунку 2.3:

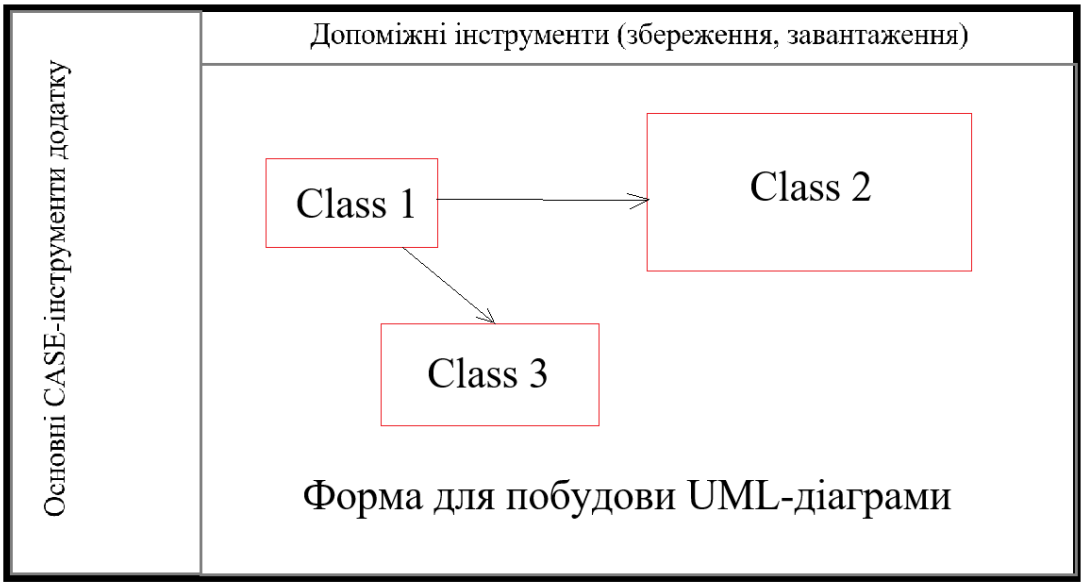

Рисунок 2.3 – Ескіз початкової форми системи

Розроблено ескіз початкової версії програмного забезпечення, що розроблюється. До форми входять такі елементи:

– основні CASE-інструменти додатку:

1) елемент діаграми класів;

2) зв'язок між класами (у вигляді прямої лінії між вершинами);

3) прибрати елемент UML-діаграми (зайвий зв'язок, вершина, метод, властивість тощо);

4) елементи інших діаграм.

– допоміжні інструменти додатку:

1) посібник користувача (якщо виникають труднощі з роботою, або є якісь питання);

2) кнопка збереження UML-діаграми, що розроблюється;

3) кнопка завантаження готової UML-діаграми, що була розроблена раніше, а також можливість її редагування;

4) налаштування системи (мова, зміна кольорової схеми);

5) завантаження готових шаблонів UML-діаграм.

– форма для побудови UML-діаграм;

– кнопки для збільшення / зменшення масштабу діаграми;

– додаткова інформація внизу форми.

2.6 Аналіз проекту

Дане програмне забезпечення являється аналогом звичайних UML-редакторів з підтримкою проектного підходу. У цьому проекті є можливість побудови діаграм та їх збереження. Проект має простий інтерфейс та інтуїтивно зрозумілий функціонал. Програма є безкоштовною, що є безпосередньою перевагою.

Розглянуто основні аналоги ринку, зібрано інформацію від опитаних респондентів у кількості 28 чоловік, завдяки чому було зроблено висновки й створено основні вимоги й задачі до цього програмного забезпечення. У розділі 3 розглянуто тестування даного програмного забезпечення й зроблено висновки на основі проведеного тестування різними методами.

2.7 Проектування динаміки системи

Результати проектування динаміки системи можуть бути представлені рядом UML-діаграм:

– діяльності (Activity Diagram);

– кооперації (Collaboration Diagram);

– послідовності (Sequence Diagram);

На рисунку 2.4 зображена діаграма послідовності розроблюваної системи:

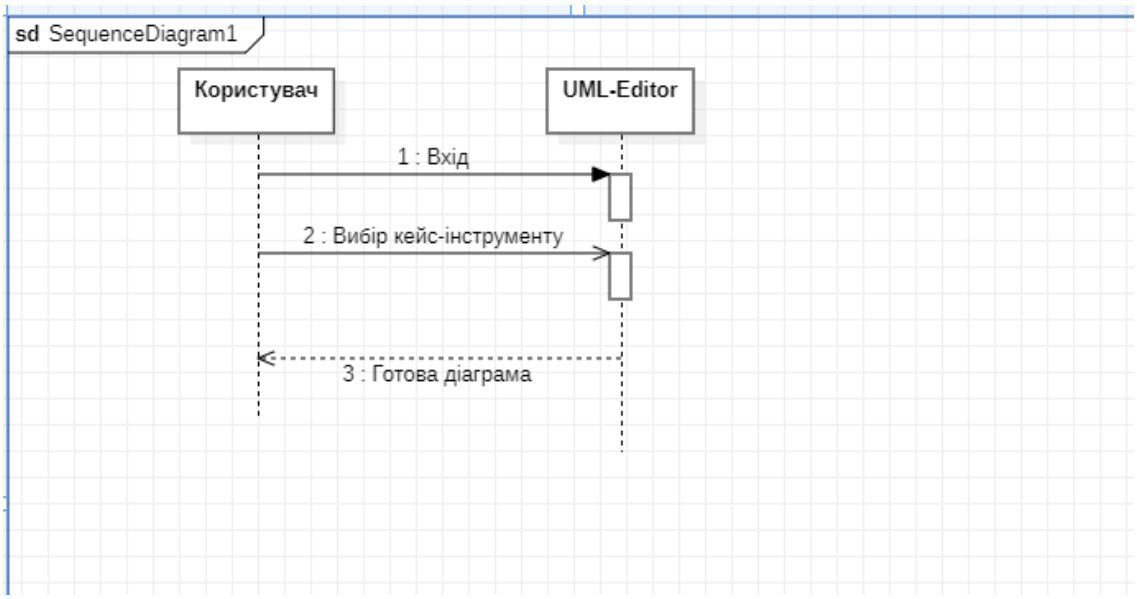

Рисунок 2.4 – Діаграма послідовності системи

У даній діаграмі послідовності можна побачити, що:

1) користувач заходить до додатку (UML-редактору): користувач звертається до програми;

2) користувач обирає кейс-інструмент для створення UML-діаграми: користувач звертається до програми;

3) UML-редактор повертає користувачу вихідні дані у вигляді побудованої діаграми: система звертається до користувача.

На рисунку 2.5 зображена діаграма діяльності розроблюваної системи.

Дана діаграма діяльності описує:

– іде запит на вхід до системи (користувач заходить до додатку);

– користувач подає запит на побудову UML-діаграми;

– система приймає запит на побудову UML-діаграми;

– користувач обирає CASE-елемент;

– система приймає запит і дає користувачу обраний CASE-елемент;

– система будує та відображає UML-діаграму.

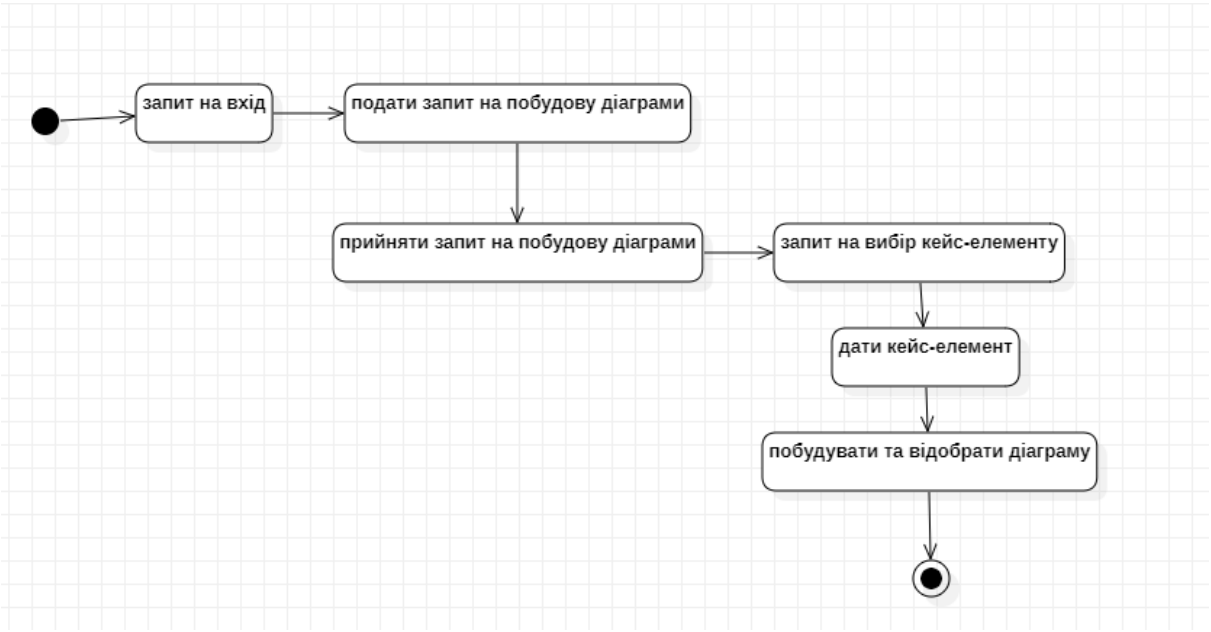

Рисунок 2.5 – Діаграма діяльності системи

На рисунку 2.6 зображена діаграма компонентів розроблюваної системи:

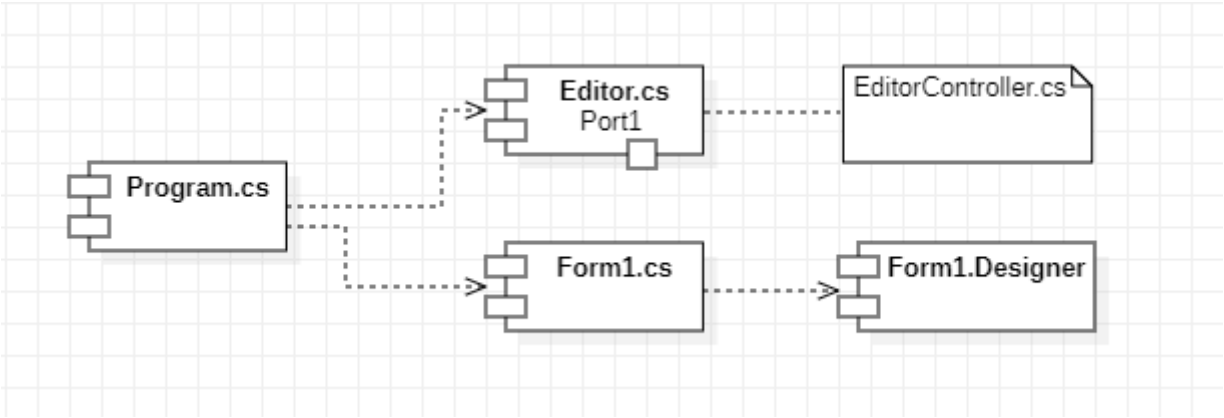

Рисунок 2.6 – Діаграма компонентів системи

Діаграма компонентів системи складається з:

– основний файл Program.cs, який містить у собі інші файли;

– файл Form1.cs, який у собі описує основні елементи форми;

– файл Form1.Designer, який містить у собі елементи форми, а також представляє форму у вигляді графічного елементу;

– файл Editor.cs, що програмно у собі описує основну логіку програми й взаємодій з інструментами CASE;

– файл EditorController.cs, що описує взаємодію з формою тощо.

2.8 Вибір мови програмування

Для розробки програмного забезпечення було обрано мову програмування C#. Це – обєктно-орієнтована мова програмування, яка є досить розпосвюдженою та популярною серед розробників та компаній у наш час. У цієї мови програмування є свої переваги, на основі яких було прийнято рішення використати цю мову у даному проекті [4].

Оскільки це – обєктно-орієнтована мова, це означає, що необхідно буде описувати абстрактні конструкції з урахуванням предметної області, а потім реалізовувати між ними взаємодію. Цей підхід користується великою популярністю, тому що дозволяє не тримати усю інформацію в голові. Мова працює на базі .NET Framework, що є безумовно значним плюсом. C# підтримує багато бібліотек, що дозволяє використовувати вже готові шаблони, а не писати свої власні, що зберігає багато часу розробнику.

Проект зроблений на основі WinForm, що дозволяє легко реалізувати графічну частину програмного забезпечення й перейти безспосередньо до логіки програми. Приклад форми розроблюваного додатку представлено на рисунку 2.7:

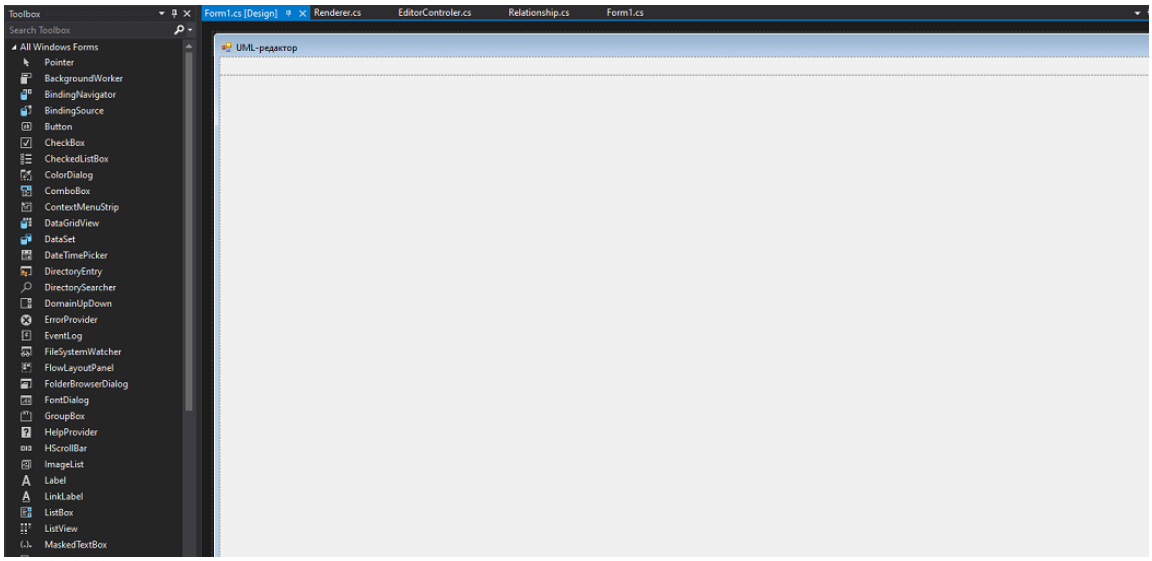

Рисунок 2.7 – Приклад WinForm

Висновки до розділу 2

У цьому розділі розробляється архітектура програми, яка краще зрозуміє функції її основних частин. Була створені різні UML-діаграми, які описували основні процеси функціонування розроблюваної системи. Описано функціональну структуру системи та її основні елементи – модулі обробки даних. Будуть визначені основні елементи системи та встановлені зв’язки між ними.

РОЗДІЛ 3 ТЕСТУВАННЯ ТА НАЛАГОДЖЕННЯ

3.1 Вибір стратегії тестування

Тестування програмного забезпечення є необхідною частиною розробки продукту. За допомогою послідовного виконання тестів можна перевірити функціональну поведінку програми, зручність використання та правильність роботи функціоналу.

Призначення тестування ПЗ є:

– знаходження та виправлення помилок до того, як продуктом почне користуватись аудиторія;

– мінімізація технічних втрат та швидкий розвиток процесу розробки через виконання тестування на декількох стадіях створення;

– оптимізація коду.

3.2 Опис тестів методами «чорного» та «білого» ящика

Тестування виконується зазвичай на декількох етапах життєвого циклу, коли розробляється програмне забезпечення. Технологія тестування програмного забезпечення містить такі етапи:

– визначення функціоналу, що підлягає та не підлягає тестуванню;

– формулювання підходів, що будуть використовуватись для даного продукту;

– написання тест кейсів;

– розробка критерію проходження тестів;

– визначення вимог середовища проведення тестування;

– проведення тестування та оцінка результатів;

– звітність результатів.

Системні вимоги:

– операційна система Windows 10 або наступних версій;

– наявність програми Microsoft Word версії 2016 або наступних;

– наявність програми Adobe Acrobat Reader DC версії 5.1 або наступних.

Апаратні вимоги:

– достатньо вільної пам’яті комп’ютері для встановлення програми;

– 32 або 64-розрядний процесор з тактовою частотою 1GHz;

– оперативна пам’ять не менше 1Gb.

Тестування проводилось на ноутбуці компанії Lenovo з процесором Intel® Core i3-6060U, тактовою частотою 2GHz, операційною системою Windows 10 та при наявності програми Microsoft Word 2016 та програми Adobe Acrobat Reader DC версії 5.1.

Перш за все, ми не маємо уявлення про структуру та внутрішній устрій системи. Потрібно зосереджуватися на тому, що програма робить, а не на тому, як вона це робить.

Оскільки це тип тестування, за визначенням може включати інші його види. Тестування чорної скриньки може бути як функціональним, так і нефункціональним. Функціональне тестування передбачає перевірку роботи функцій системи, а нефункціональне відповідно загальні характеристики нашої програми.

Техніка чорного ящика застосовна в усіх рівнях тестування (від модульного до приймального), котрим існує специфікація. Наприклад, при здійсненні системного або інтеграційного тестування вимоги, або функціональна специфікація будуть основою для написання тест-кейсів.

Техніки тест-дизайну, засновані на використання чорної скриньки, включають:

– класи еквівалентності;

– аналіз граничних значень;

– таблиці рішень;

– діаграми зміни стану;

– тестування всіх пар.

Переваги:

– тестування проводиться з позиції кінцевого користувача і може допомогти виявити неточності та протиріччя у специфікації;

– тестувальнику не потрібно знати мови програмування та заглиблюватися особливо реалізації програми;

– тестування може проводитись фахівцями, незалежними від відділу розробки, що допомагає уникнути упередженого ставлення;

– можна починати писати тест-кейси, як тільки готова специфікація.

Недоліки:

– тестується лише дуже обмежена кількість шляхів виконання програми;

– без чіткої специфікації (а це скоріше реальність на багатьох проектах) досить складно скласти ефективні тест-кейси;

– деякі тести можуть бути надмірними, якщо вони вже були проведені розробником на рівні модульного тестування;

Протилежністю техніки чорної скриньки є тестування методом білої скриньки, про яку йдеться нижче.

Тестування методом білої скриньки (також: прозорої, відкритої, скляної скриньки; засноване на коді або структурне тестування) – метод тестування програмного забезпечення, який передбачає, що внутрішня структура/пристрій/реалізація системи відома тестувальнику. Ми обираємо вхідні значення, ґрунтуючись на знанні коду, який їх оброблятиме. Так само ми знаємо, яким має бути результат цієї обробки. Знання всіх особливостей тестованої програми та її реалізації обов'язкові для цієї техніки. Тестування білої скриньки – заглиблення у внутрішній пристрій системи, межі її зовнішніх інтерфейсів.

Відповідно до ISTQB:

Тестування методом білої скриньки – це:

– тестування, засноване на аналізі внутрішньої структури компонента або системи.

– тест-дизайн, що базується на техніці білої скриньки – процедура написання або вибору тест-кейсів на основі аналізу внутрішнього пристрою системи або компонента.

Техніка білого ящика застосовна різних рівнях тестування – від модульного до системного, але переважно застосовується саме реалізації модульного тестування компонента його автором.

Переваги:

– тестування може проводитися на ранніх етапах: немає необхідності чекати створення інтерфейсу користувача;

– можна провести ретельніше тестування, з покриттям великої кількості шляхів виконання програми.

Недоліки:

– для виконання тестування білої скриньки необхідна велика кількість спеціальних знань

– під час використання автоматизації тестування на цьому рівні, підтримка тестових скриптів може бути досить накладною, якщо програма часто змінюється.

Під час тестування даного програмного забезпечення було проведено перевірку усіх функціональних вимог та варіантів використання, враховано роботу інтерфейсу та перевірено усі запобіжні функції від некоректних дій користувача. Результати тестів наведені в таких таблицях:

– завантаження CASE-засобу (таблиця 3.1);

– перевірка роботи при не вказаних властивостях (таблиця 3.2);

– побудова діаграми CASE-елементами даними в системі (таблиця 3.3);

– перевірка роботи порівняння відсутності одного з елементів (таблиця 3.4);

– збереження інформації (таблиця 3.5).

Таблиця 3.1 – Завантаження CASE-засобу

Мета тесту	Перевірити роботу функції завантаження CASE-елементів з системи.
Початковий стан	Програма запущена. Видно інтерфейс користувача.
Вхідні дані	-
Проведення тесту	Натиснути кнопку для першого файлу, обрати потрібний елемент з системи, натиснути кнопку «Відкрити».
Очікуваний результат	В обраному полі вибереться потрібний елемент і буде готовий до роботи.
Фактичний результат	В обраному полі вибереться потрібний елемент і буде готовий до роботи.

Ціль тесту: у даному тесті перевірявся процес завантаження CASE-засобу з форми. Це допоможе уникнути помилок під час проектування.

Процес тестування: після завантаження основної форми програмного забезпечення, обирається довільний CASE-елемент й перевіряється правильність його роботи.

Результати тесту: очікуваний результат збігається з фактичним. Результат тестування можна вважати успішним.

Висновки: після успішного завершення тестування правильності роботи CASE-елементів, можна переходити до наступних тестувань додатку.

Таблиця 2.2 – Перевірка роботи при не вказаних властивостях документа

Мета тесту	Перевірити роботу виведення повідомлення про некоректну роботу CASE-елементу через відсутність параметрів.
Початковий стан	Обрано CASE-елемент з системи.
Вхідні дані	CASE-елемент.
Проведення тесту	Натиснути кнопку потрібного CASE-елементу.
Очікуваний результат	Продовження роботи з обраним CASE-елементом.
Фактичний результат	Потім вивелося інше повідомлення «Оберіть CASE-елемент»

Ціль тесту: перевірити роботу програми, якщо у CASE-елементу відсутні параметри.

Процес тестування: обрано CASE-елемент з відсутніми параметрами, після чого перевіряється робота додатку при продовженні роботи з ним.

Результати тесту: програма некоректно обробляла такі запити, користувач й надалі мав можливість працювати з такими CASE-елементами.

Висновки: після успішного завершення тестування, було розроблено додаткове повідомлення про помилку й попередження, що потрібно обирати CASE-елементи з усіма параметрами.

Таблиця 3.3 – Побудова діаграми CASE-елементами даними в системі

Мета тесту	Перевірити роботу функції побудови UML-діаграми та точність її результату.
Початковий стан	Обрати UML-діаграму, яка буде побудована.
Вхідні дані	Назва файлу, тип UML-діаграми.
Проведення тесту	Натиснути кнопку потрібного CASE-елементу.
Очікуваний результат	В окремому полі відображається обраний CASE-елемент. Продовження роботи.
Фактичний результат	В окремому полі відображається обраний CASE-елемент. Продовження роботи.

Ціль тесту: перевірити правильність побудови UML-діаграми за допомогою використання CASE-елементів.

Процес тестування: обирається потрібний CASE-елемент й виводиться на форму, після чого на формі відображається елемент UML-діаграми.

Результати тесту: очікуваний результат збігається з фактичним. Результат тестування можна вважати успішним.

Висновки: після успішного завершення тестування правильності роботи CASE-елементів для побудови UML-діаграми, можна переходити до наступних тестувань додатку.

Таблиця 3.4 – Перевірка роботи порівняння відсутності одного з елементів

Мета тесту	Перевірити роботу виведення помилки при некоректному зверненні.
Початковий стан	Обраний потрібний CASE-елемент. Обраний тип діаграми.
Вхідні дані	CASE-елементи обраної діаграми.
Проведення тесту	Натиснути кнопку елементу.
Очікуваний результат	У додатковому вікні виведеться зображення потрібного CASE-елементу.
Фактичний результат	У додатковому вікні виведеться зображення потрібного CASE-елементу.

Ціль тесту: перевірити коректність роботи додатку при неправильному зверненні до CASE-елементу.

Процес тестування: перевіряються будь-які спроби звернення до CASE-елементів додатку, щоб уникнути ситуацій збою програми через неправильне звернення.

Результати тесту: очікуваний результат збігається з фактичним. Результат тестування можна вважати успішним.

Висновки: після успішного завершення тестування коректності роботи CASE-елементів при неправильному зверненні, можна переходити до наступних тестувань додатку.

Таблиця 3.5 – Збереження інформації

Мета тесту	Перевірити функцію збереження результатів.
Початковий стан	Виведені результати побудованої UML-діаграми та виділені символи.
Вхідні дані	Результати побудови обраної UML-діаграми
Проведення тесту	Натиснути кнопку «Зберегти».
Очікуваний результат	Створиться файл та запишуться усі результати.
Фактичний результат	Створився файл та усі результати збереглися.

Ціль тесту: перевірити програму на функцію збереження результатів побудованих UML-діаграм.

Процес тестування: після побудови UML-діаграми, починається процес збереження результату у окремому файлі.

Результати тесту: очікуваний результат збігається з фактичним. Результат тестування можна вважати успішним.

Висновки: успішно завершено тестування збереження результатів побудованих UML-діаграм.

3.3 Аналіз помилок, їх влив на систему і вирішення проблеми

Досліджено аналіз якості програмного забезпечення. Виявлено, що для даного продукту потрібно протестувати роботу інтерфейсу користувача та функціональні вимоги.

Окрім цього, вказано опис процесу тестування, сформульовані системні та апаратні вимоги, що є необхідними для проведення даного етапу розробки продукту. Наведено вимоги до технічного та програмного забезпечення, на якому будуть проводитися тести.

Також детально описані проведені тести-кейси у вигляді таблиць. Вказано інформацію про мету тестування, початковий стан програми, вхідні дані, вказані кроки проведення тестів, а також очікуваний та фактичний результат. Проводилось тестування усіх функціональних вимог програмного забезпечення та робота виведення повідомлень на некоректні дії користувача.

Висновки до розділу 3

Процес тестування та налагодження є одним з основних етапів розробки програмного забезпечення. Тестування потрібно проводити на усіх етапах розробки. Це допомагає усувати помилки вже на початкових етапах й більш детально розуміти, які проблему можуть виникнути під час переходу до інших етапів, які так само потрібно тестувати після завершення роботи над ними.

Кінець розробки програмного забезпечення означає, що потрібно переходити до повного тестування роботи програми, щоб переконатися у правильності роботи усього функціоналу, взаємодії між різними функціями тощо.

ВИСНОВКИ

Однією з найперших задач під час виконання бакалаврської роботи являється постановка задачі та збір вимог до програмного забезпечення. Для виконання цих етапів було проведено опитування зацікавлених сторін у вигляді 28 респондентів з різних сфер діяльності, що допомогло покращити розуміння основних та додаткових задач.

Наступним етапом було проектування й визначення основних специфікацій програми. Завдяки отриманій інформації під час збору вимог, було створено основні діаграми для подальшої розробки проекту. На основі цих діаграм створено додаток для побудови UML-діаграм.

Завершальним етапом розробки стало тестування та налагодження. Було проведено основні тести, які допомогли виявити основні недоліки та баги програми. Тестування проводилося методами білого та чорного ящиками, що мають різні переваги та недоліки. Після цих тестів виправлено роботу програми.

Створено CASE-проект, який являється досить актуальним у наш час, оскільки більшість розробників та компаній користуються схожими програмами й сайтами для проектування свого програмного забезпечення. UML-діаграми допомагають якісно оцінити проект й розставити пріоритети під час подальшої розробки та кодування. Проектний підхід значно прискорює процес побудови UML-діаграм, що є досить вагомою перевагою під час вибору програми для проектування свого програмного забезпечення.

СПИСОК ЛІТЕРАТУРИ

1. Інженерія програмування для дизайну програмного забезпечення безпеки. - М .: Радіо і зв'язок, 1985, - 512с.

2. Вендров А. М. CASE-технології. Сучасні методи та засоби проектування інформаційних систем / А. М. Вендров. - М.: Фінанси та статистика, 1998. - 176 с.

3. Mayo D. Microsoft Visual Studio 2010 Self-tutorial = Microsoft Visual Studio 2010: Посібник для початківців. – С.: «БХВ-Петербург», 2010. – С. 464. – ISBN 978-5-9775-0609-0

4. Alex McKee Представляємо .NET 4.0 і Visual Studio 2010 для професіоналів = Представляємо .NET 4.0: з Visual Studio 2010. - М.:
Вільямс, 2010. – С. 416. – ISBN 978-5-8459-1639-6

ДОДАТКИ

ТЕКСТ ПРОГРАМИ

BaseGenerator.cs:

```
namespace UML_Editor.CodeGeneration
{
   public abstract class BaseGenerator
   {
      public virtual List<string> Lines { get; set; } = new List<string>();

      public virtual void InsertLines(string file)
      {
         using (StreamWriter sw = File.AppendText(file))
         {
            foreach (string line in Lines)
            {
               sw.WriteLine(line);
            }
            sw.WriteLine();
         }
      }
   }
}
```

ClassGenerator.cs:

```
using UML_Editor.ProjectStructure;

namespace UML_Editor.CodeGeneration
{
   public class ClassGenerator : BaseGenerator
   {
      public ClassStructure ClassStructure { get; set; }
      public ConstructorGenerator Constructor { get; set; }
      public List<PropertyGenerator> Properties { get; set; }
      public List<MethodGenerator> Methods { get; set; }

      public ClassGenerator(ClassStructure structure)
      {
         ClassStructure = structure;
         Constructor = new ConstructorGenerator(structure, structure.Properties);
         Methods = new List<MethodGenerator>();
         foreach (MethodStructure method in structure.Methods)
            Methods.Add(new MethodGenerator(method));
         Properties = new List<PropertyGenerator>();
         foreach (PropertyStructure prop in structure.Properties)
            Properties.Add(new PropertyGenerator(prop));
      }

      public override void InsertLines(string file)
      {
         string line = "    public class " + ClassStructure.Name;
         using (StreamWriter sw = File.CreateText(file))
         {
            sw.WriteLine("using System;");
            sw.WriteLine();
            sw.WriteLine("namespace TestingGen");
            sw.WriteLine("{");
            sw.WriteLine(line);
            sw.WriteLine("    {");
         }
         Properties.ForEach(x => x.InsertLines(file));
         Constructor.InsertLines(file);
         Methods.ForEach(x => x.InsertLines(file));
         using (StreamWriter sw = File.AppendText(file))
         {
            sw.WriteLine("    }");
            sw.WriteLine("}");
         }
      }
   }
}
```

CodeGenerator.cs:

```
using UML_Editor.ProjectStructure;

namespace UML_Editor.CodeGeneration
```

```
{
    public class CodeGenerator
    {
        public Project Project { get; set; }
        public string DirectoryPath { get; set; }
        List<ClassGenerator> Classes = new List<ClassGenerator>();

        public CodeGenerator(Project project, string dir)
        {
            Project = project;
            DirectoryPath = dir;
            if (!Directory.Exists(dir))
            {
                Directory.CreateDirectory(dir);
            }
            foreach (ClassStructure klass in Project.Classes)
            {
                Classes.Add(new ClassGenerator(klass));
            }
        }

        public void Generate()
        {
            foreach (ClassGenerator gen in Classes)
            {
                string file = DirectoryPath + "\\" + gen.ClassStructure.Name + ".cs";
                gen.InsertLines(file);
            }
        }
    }
}
```

ConstructorGenerator.cs:

```
using UML_Editor.ProjectStructure;

namespace UML_Editor.CodeGeneration
{
    public class ConstructorGenerator : BaseGenerator
    {
        public List<PropertyStructure> Properties { get; set; }
        public ClassStructure Class { get; set; }
        public ConstructorGenerator(ClassStructure klass, List<PropertyStructure> props)
        {
            Properties = props;
            Class = klass;
            GenerateLines();
        }

        private void GenerateLines()
        {
            string Arguments = "";
            foreach (PropertyStructure prop in Properties)
            {
                Arguments += prop.Type + " ";
                Arguments += prop.Name.ToLower() + ", ";
            }

            if (Arguments.Length > 1)
                Arguments = Arguments.Substring(0, Arguments.Length - 2);
            string line = "        public " + Class.Name + "(" + Arguments + ")";
            Lines.Add(line);
            Lines.Add("        {");
            foreach (PropertyStructure prop in Properties)
            {
                Lines.Add("            " + prop.Name + " = " + prop.Name.ToLower() + ";");
            }
            Lines.Add("        }");
        }
    }
}
```

MethodGenerator.cs:

```
using UML_Editor.ProjectStructure;

namespace UML_Editor.CodeGeneration
{
    public class MethodGenerator : BaseGenerator
    {
        public MethodStructure Structure { get; set; }
        public MethodGenerator(MethodStructure structure)
```

```
        {
            Structure = structure;
            GenerateLines();
        }

        private void GenerateLines()
        {
            string Arguments = "";
            List<string> parts = Structure.Arguments.Split(',').ToList();
            List<string> names = parts.Select(x => Regex.Replace(x.Split(':')[0], @"\s+", "")).ToList();
            List<string> types = parts.Select(x => Regex.Replace(x.Split(':')[1], @"\s+", "")).ToList();
            for (int i = 0; i < names.Count; i++)
            {
                Arguments += types[0] + " ";
                Arguments += names[0] + ", ";
            }
            if (Arguments.Length > 1)
                Arguments = Arguments.Substring(0, Arguments.Length - 2);
            string line = "        public " + Structure.Type + " " + Structure.Name + "(" + Arguments + ")";
            Lines.Add(line);
            Lines.Add("        {");
            Lines.Add("        }");
        }
    }
}
```

PropertyGenerator.cs:

```
using UML_Editor.ProjectStructure;

namespace UML_Editor.CodeGeneration
{
    public class PropertyGenerator : BaseGenerator
    {
        public PropertyStructure Structure { get; set; }

        public PropertyGenerator(PropertyStructure structure)
        {
            Structure = structure;
            GenerateLines();
        }

        private void GenerateLines()
        {
            string line = "        public " + Structure.Type + " " + Structure.Name + " { get; set; }";
            Lines.Add(line);
        }
    }
}
```

AccessModifiers.cs:

```
namespace UML_Editor.Enums
{
    public enum AccessModifiers
    {
        Private,
        Public,
        Protected
    }
}
```

Modifiers.cs:

```
namespace UML_Editor.Enums
{
    public enum Modifiers
    {
        None,
        Static,
        Abstract
    }
}
```

RelationshipType.cs:

```
namespace UML_Editor.Enums
{
    public enum RelationshipType
    {
    }
}
```

CodeStructureEventArgs.cs:

```
using UML_Editor.ProjectStructure;

namespace UML_Editor.EventArguments
{
   public class CodeStructureEventArgs : EventArgs
   {
      public BasicCodeStructure CodeStructure { get; }

      public CodeStructureEventArgs(BasicCodeStructure structure)
      {
         CodeStructure = structure;
      }
   }
}
```

EditorChangeEventArgs.cs:

```
namespace UML_Editor
{
   public class EditorChangeEventArgs
   {
      public int Index;
      public EditorChangeEventArgs(int index)
      {
         Index = index;
      }
   }
}
```

HitboxEventArgs.cs:

```
using UML_Editor.Hitboxes;

namespace UML_Editor
{
   public class HitboxEventArgs : EventArgs
   {
      public IHitbox Hitbox;

      public HitboxEventArgs(IHitbox hitbox)
      {
         Hitbox = hitbox;
      }
   }
}
```

NodeEventArgs.cs:

```
using UML_Editor.Nodes;

namespace UML_Editor.EventArguments
{
   public class NodeEventArgs : EventArgs
   {
      public INode Node { get; }

      public NodeEventArgs(INode node)
      {
         Node = node;
      }
   }
}
```

OptionsMenuEventArgs.cs:

```
using UML_Editor.Nodes;

namespace UML_Editor.EventArguments
{
   public class OptionsMenuEventArgs : EventArgs
   {
      public IOptionsNode Node { get; private set; }

      public OptionsMenuEventArgs(IOptionsNode optionsNode)
      {
         Node = optionsNode;
      }
   }
}
```

PositionEventArgs.cs:

```
using UML_Editor.Geometry;
using UML_Editor.Rendering;

namespace UML_Editor
{
```

```
    public class PositionEventArgs : EventArgs
    {
        public Vector Position;
        public PositionEventArgs(Vector position)
        {
            Position = position;
        }
    }
}
```

ResizeEventArgs.cs:

```
namespace UML_Editor
{
    public class ResizeEventArgs : EventArgs
    {
        public float Width { get; }
        public float Height { get; }
        public ResizeEventArgs(float width, float height)
        {
            Width = width;
            Height = height;
        }
    }
}
```

TextEventArgs.cs:

```
namespace UML_Editor.EventArguments
{
    public class TextEventArgs : EventArgs
    {
        public string Text { get; }

        public TextEventArgs(string text)
        {
            Text = text;
        }
    }
}
```

Line.cs:

```
using UML_Editor.Rendering;

namespace UML_Editor.Geometry
{
    public struct Line
    {
        public Vector StartPoint { get; set; }
        public Vector EndPoint { get; set; }
        public Line(Vector start, Vector end)
        {
            StartPoint = start;
            EndPoint = end;
        }
    }
}
```

Šafránková.cs:

```
namespace UML_Editor.Geometry
{
    public static class Šafránková
    {
        public static bool Intersect(Line l1, Line l2)
        {
            var dir1 = Direction(l1.StartPoint, l1.EndPoint, l2.StartPoint);
            var dir2 = Direction(l1.StartPoint, l1.EndPoint, l2.EndPoint);
            var dir3 = Direction(l2.StartPoint, l2.EndPoint, l1.StartPoint);
            var dir4 = Direction(l2.StartPoint, l2.EndPoint, l1.EndPoint);

            if (dir1 != dir2 && dir3 != dir4) return true;
            return false;
        }

        private static LineDirection Direction(Vector a, Vector b, Vector c)
        {
            float value = (b.Y - a.Y) * (c.X - b.X) - (b.X - a.X) * (c.Y - b.Y);

            if (value == 0) return LineDirection.Collinear;
            else if (value < 0) return LineDirection.AntiClockwise;
```

```
            return LineDirection.Clockwise;
        }


        private enum LineDirection
        {
            Collinear,
            Clockwise,
            AntiClockwise
        }
    }
}
```

Vector.cs:

```
namespace UML_Editor.Geometry
{
    public struct Vector
    {
        public float X { get; set; }
        public float Y { get; set; }

        public Vector(float x, float y)
        {
            X = x;
            Y = y;
        }

        public static Vector Zero = new Vector(0, 0);

        public static Vector operator +(Vector left, Vector right)
        {
            return new Vector(left.X + right.X, left.Y + right.Y);
        }
        public static Vector operator -(Vector left, Vector right)
        {
            return new Vector(left.X - right.X, left.Y - right.Y);
        }
        public static Vector operator *(Vector left, Vector right)
        {
            return new Vector(left.X * right.X, left.Y * right.Y);
        }
        public static Vector operator /(Vector left, Vector right)
        {
            return new Vector(left.X / right.X, left.Y / right.Y);
        }
        public static bool operator ==(Vector left, Vector right)
        {
            return left.X == right.X && left.Y == right.Y;
        }
        public static bool operator !=(Vector left, Vector right)
        {
            return left.X != right.X || left.Y != right.Y;
        }

        public static implicit operator Point(Vector v)
        {
            return new Point((int)v.X, (int)v.Y);
        }
        public static implicit operator Vector(Point p)
        {
            return new Vector(p.X, p.Y);
        }

        public static implicit operator PointF(Vector v)
        {
            return new PointF(v.X, v.Y);
        }
        public static implicit operator Vector(PointF p)
        {
            return new Vector(p.X, p.Y);
        }
        public static Vector operator +(Vector left, int right)
        {
            return new Vector(left.X + right, left.Y + right);
        }
        public static Vector operator -(Vector left, int right)
        {
            return new Vector(left.X - right, left.Y - right);
```

```
        }
        public static Vector operator *(Vector left, int right)
        {
            return new Vector(left.X * right, left.Y * right);
        }
        public static Vector operator /(Vector left, int right)
        {
            return new Vector(left.X / right, left.Y / right);
        }
        public static Vector operator +(Vector left, float right)
        {
            return new Vector(left.X + right, left.Y + right);
        }
        public static Vector operator -(Vector left, float right)
        {
            return new Vector(left.X - right, left.Y - right);
        }
        public static Vector operator *(Vector left, float right)
        {
            return new Vector(left.X * right, left.Y * right);
        }
        public static Vector operator /(Vector left, float right)
        {
            return new Vector(left.X / right, left.Y / right);
        }
        public static int GetDistance(Vector v)
        {
            return (int)Math.Sqrt((int)Math.Pow((double)v.X, 2) + (int)Math.Pow((double)v.Y, 2));
        }
    }
}
```

IHitbox.cs:

```
using UML_Editor.Geometry;
using UML_Editor.Rendering;

namespace UML_Editor.Hitboxes
{
    public interface IHitbox
    {
        bool HasTriggered(Vector position);
        bool IsTriggerable { get; set; }
    }
}
```

RectangleHitbox.cs:

```
using UML_Editor.Geometry;
using UML_Editor.Rendering;

namespace UML_Editor.Hitboxes
{
    public class RectangleHitbox : IHitbox
    {
        public RectangleHitbox(Vector position, float width, float height)
        {
            Position = new Vector(position.X, position.Y);
            Width = width;
            Height = height;
        }

        public float Width { get; set; }
        public float Height { get; set; }
        public Vector Position { get; set; }
        public bool IsTriggerable { get; set; } = true;

        public bool HasTriggered(Vector position)
        {
            float left = Position.X;
            float right = Position.X + Width;
            float top = Position.Y;
            float bot = Position.Y + Height;
            return position.X <= right && position.X >= left && position.Y <= bot && position.Y >= top;
        }

        public static RectangleHitbox CreateFromLine(Vector start, Vector end, int width)
        {
            Vector pos = Vector.Zero;
            int height = 0;
            if(start.Y < end.Y)
            {
```

```
                pos.Y = start.Y;
                pos.X = start.X - (width / 2);
                height = Vector.GetDistance(start - end);
                return new RectangleHitbox(pos, width, height);
            }
            else if(start.Y > end.Y)
            {
                pos.Y = end.Y;
                pos.X = end.X - (width / 2);
                height = Vector.GetDistance(start - end);
                return new RectangleHitbox(pos, width, height);
            }
            else if(start.X < end.X)
            {
                pos.X = start.X;
                pos.Y = start.Y - (width / 2);
                height = Vector.GetDistance(start - end);
                return new RectangleHitbox(pos, height, width);
            }
            else
            {
                pos.X = end.X;
                pos.Y = end.Y - (width / 2);
                height = Vector.GetDistance(start - end);
                return new RectangleHitbox(pos, height, width);
            }
        }
    }
}
```

IContainerNode.cs:

```
using UML_Editor.EventArguments;

namespace UML_Editor.Nodes
{
    public interface IContainerNode : INode
    {
        List<INode> Children { get; set; }
        List<IFocusableNode> GetFocusableNodes();
        IFocusableNode FocusedNode { get; set; }
        EventHandler<NodeEventArgs> OnNodeAdd { get; set; }
        EventHandler<NodeEventArgs> OnNodeRemoval { get; set; }
    }
}
```

IFocusableNode.cs:

```
using UML_Editor.EventArguments;

namespace UML_Editor.Nodes
{
    public interface IFocusableNode : IRenderableNode
    {
        EventHandler<NodeEventArgs> OnFocused { get; set; }
        EventHandler<NodeEventArgs> OnUnfocused { get; set; }
    }
}
```

IKeyboardHandlerNode.cs:

```
namespace UML_Editor.Nodes
{
    public interface IKeyboardFocusableNode : IFocusableNode
    {
        EventHandler<KeyPressEventArgs> OnKeyPress { get; set; }
    }
}
```

IMouseHandlerNode.cs:

```
using UML_Editor.Hitboxes;

namespace UML_Editor.Nodes
{
    public interface IMouseFocusableNode : IFocusableNode
    {
        EventHandler OnMouseClick { get; set; }
    }
}
```

INode.cs:

```
using UML_Editor.Enums;
using UML_Editor.EventArguments;
using UML_Editor.Geometry;
using UML_Editor.Hitboxes;
```

```
using UML_Editor.Rendering;
using UML_Editor.Rendering.RenderingElements;

namespace UML_Editor.Nodes
{
    public interface INode
    {
        Vector Position { get; set; }
        float Width { get; set; }
        float Height { get; set; }
        List<IHitbox> TriggerAreas { get; set; }
        EventHandler<PositionEventArgs> OnPositionChanged { get; set; }
        EventHandler<ResizeEventArgs> OnResize { get; set; }
        EventHandler OnChange { get; set; }
        EventHandler<NodeEventArgs> OnRemoval { get; set; }
    }
}
```

IOptionsNode.cs:

```
using UML_Editor.Nodes;

namespace UML_Editor.Nodes
{
    public interface IOptionsNode : IMouseFocusableNode
    {
        BasicContainerNode OptionsPrefab { get; set; }
        BasicContainerNode OptionsMenu { get; set; }
        EventHandler OnOptionsShow { get; set; }
        EventHandler OnOptionsHide { get; set; }
    }
}
```

IRenderableNode.cs:

```
using UML_Editor.Rendering.RenderingElements;
using UML_Editor.Rendering;

namespace UML_Editor.Nodes
{
    public interface IRenderableNode : INode
    {
        void Render(Renderer renderer);
    }
}
```

ITextNode.cs:

```
using UML_Editor.EventArguments;

namespace UML_Editor.Nodes
{
    public interface ITextNode : INode
    {
        string Text { get; set; }
        EventHandler<TextEventArgs> OnTextChange { get; set; }
    }
}
```

BasicContainerNode.cs:

```
using UML_Editor.EventArguments;
using UML_Editor.Geometry;
using UML_Editor.NodeStructure;
using UML_Editor.Rendering;
using UML_Editor.Rendering.ElementStyles;

namespace UML_Editor.Nodes
{
    public class BasicContainerNode : BasicNode, IContainerNode
    {
        public override Vector Position
        {
            get => base.Position;
            set
            {
                base.Position = value;
                RepositionChildren();
            }

        }

        public BasicContainerNode(BasicNodeStructure structure, RectangleRenderElementStyle border_style) : base(structure,
border_style)
        {
        }
```

```
        public virtual void RepositionChildren()
        {
            for (int i = 0; i < Children.Count; i++)
            {
                Children[i].Position = new Vector(Position.X, Position.Y + i * Renderer.SingleTextHeight);
            }
        }

        public virtual void AddNode(INode node)
        {
            Children.Add(node);
            if (node is IFocusableNode fn)
            {
                fn.OnFocused += OnNodeFocus;
                fn.OnUnfocused += OnNodeUnfocus;
            }
            node.OnResize += OnChildResize;
            RepositionChildren();
            OnNodeAdd?.Invoke(this, new NodeEventArgs(node));
        }
        public virtual void RemoveNode(INode node)
        {
            if(Children.Contains(node))
            {
                Children.Remove(node);
                RepositionChildren();
                OnNodeRemoval?.Invoke(this, new NodeEventArgs(node));
            }
        }
        public virtual void RemoveNode(int index)
        {
            if (index >= 0 && index < Children.Count)
            {
                INode removed = Children[index];
                Children.RemoveAt(index);
                OnNodeRemoval?.Invoke(this, new NodeEventArgs(removed));
            }
        }
        public virtual void OnNodeFocus(object sender, NodeEventArgs e)
        {
            if (FocusedNode != e.Node)
            {
                FocusedNode?.OnUnfocused?.Invoke(this, new NodeEventArgs(FocusedNode));
                FocusedNode = (IFocusableNode) e.Node;
            }
        }
        public virtual void OnNodeUnfocus(object sender, NodeEventArgs e)
        {
            if (FocusedNode == e.Node)
                FocusedNode = null;
        }

        public virtual void OnChildResize(object sender, ResizeEventArgs e)
        {
            RepositionChildren();
        }

        public override void Render(Renderer renderer)
        {
            base.Render(renderer);
            Children.OfType<IRenderableNode>().ToList().ForEach(x => x.Render(renderer));
        }

        public virtual void PrependNode(INode node)
        {
            Children.Insert(0, node);
        }

        public List<IFocusableNode> GetFocusableNodes() => Children.OfType<IFocusableNode>().ToList();
        public virtual List<INode> Children { get; set; } = new List<INode>();
        public IFocusableNode FocusedNode { get; set; }
        public EventHandler<NodeEventArgs> OnNodeAdd { get; set; }
        public EventHandler<NodeEventArgs> OnNodeRemoval { get; set; }
    }
}
```

BasicNode.cs:

```
using System.Threading.Tasks;
```

```
using UML_Editor.EventArguments;
using UML_Editor.Geometry;
using UML_Editor.Hitboxes;
using UML_Editor.NodeStructure;
using UML_Editor.Rendering;
using UML_Editor.Rendering.ElementStyles;
using UML_Editor.Rendering.RenderingElements;

namespace UML_Editor.Nodes
{
    public class BasicNode : IRenderableNode
    {
        public virtual BasicNodeStructure Structure { get; }
        public RectangleRenderElementStyle BorderStyle { get; }
        public List<IHitbox> TriggerAreas { get; set; } = new List<IHitbox>();

        public RectangleRenderElement BorderElement { get; set; }
        public EventHandler<ResizeEventArgs> OnResize { get; set; }
        public EventHandler OnChange { get; set; }
        public EventHandler<NodeEventArgs> OnRemoval { get; set; }
        public EventHandler<PositionEventArgs> OnPositionChanged { get; set; }
        public BasicNode(BasicNodeStructure structure, RectangleRenderElementStyle border_style)
        {
            Structure = structure;
            BorderStyle = border_style;
            BorderElement = new RectangleRenderElement(Position, Width, Height, border_style.FillColor, border_style.BorderColor,
border_style.BorderWidth);
            TriggerAreas.Add(new RectangleHitbox(Position, Width, Height));
        }
        public virtual void Render(Renderer renderer)
        {
            BorderElement.Render(renderer);
        }
        public virtual Vector Position
        {
            get => Structure.Position;
            set
            {
                Structure.Position = value;
                BorderElement.Position = value;
                ((RectangleHitbox)TriggerAreas[0]).Position = value;
                OnPositionChanged?.Invoke(this, new PositionEventArgs(value));
                OnChange?.Invoke(this, EventArgs.Empty);
            }
        }
        public virtual float Width
        {
            get => Structure.Width;
            set
            {
                Structure.Width = value;
                BorderElement.Width = value;
                ((RectangleHitbox)TriggerAreas[0]).Width = value;
                OnResize?.Invoke(this, new ResizeEventArgs(Width, Height));
                OnChange?.Invoke(this, EventArgs.Empty);
            }
        }
        public virtual float Height
        {
            get => Structure.Height;
            set
            {
                Structure.Height = value;
                BorderElement.Height = value;
                ((RectangleHitbox)TriggerAreas[0]).Height = value;
                OnResize?.Invoke(this, new ResizeEventArgs(Width, Height));
                OnChange?.Invoke(this, EventArgs.Empty);
            }
        }
        public virtual Color BorderColor
        {
            get => BorderStyle.BorderColor;
            set
            {
                BorderStyle.BorderColor = value;
                BorderElement.BorderColor = value;
                OnChange?.Invoke(this, EventArgs.Empty);
            }
```

```
            }
            public virtual Color FillColor
            {
                get => BorderStyle.FillColor;
                set
                {
                    BorderStyle.FillColor = value;
                    BorderElement.FillColor = value;
                    OnChange?.Invoke(this, EventArgs.Empty);
                }
            }
        }
    }
```

ButtonNode.cs:

```
using UML_Editor.Rendering;
using UML_Editor.Rendering.RenderingElements;
using UML_Editor.Rendering.ElementStyles;
using System.Drawing;
using UML_Editor.Hitboxes;
using UML_Editor.Geometry;
using System.Windows.Forms;
using UML_Editor.EventArguments;
using UML_Editor.NodeStructure;

namespace UML_Editor.Nodes
{
    public class ButtonNode : IRenderableNode, IMouseFocusableNode, ITextNode
    {
        public ButtonStructure Structure { get; }
        public RectangleRenderElementStyle BorderStyle { get; }
        public TextRenderElementStyle TextStyle { get; }

        public ButtonNode(ButtonStructure structure, RectangleRenderElementStyle style, TextRenderElementStyle text_style)
        {
            Structure = structure;
            BorderStyle = style;
            TextStyle = text_style;
            BorderElement = new RectangleRenderElement(Position, Width, Height, style.FillColor, style.BorderColor,
style.BorderWidth);
            TextElement = new TextRenderElement(Position, Text, TextColor, TextStyle.FontSize, TextStyle.FontStyle);
            TriggerAreas.Add(new RectangleHitbox(Position, Width, Height));
            ButtonAction = Structure.ButtonAction;
            OnMouseClick += (object sender, EventArgs e) =>
            {
                ButtonAction.Invoke();
                OnUnfocused?.Invoke(this, new NodeEventArgs(this));
            };
        }

        public List<IHitbox> TriggerAreas { get; set; } = new List<IHitbox>();
        public EventHandler<PositionEventArgs> OnPositionChanged { get; set; }

        public Vector Position
        {
            get => Structure.Position;
            set
            {
                Structure.Position = value;
                BorderElement.Position = value;
                TextElement.Position = value;
                ((RectangleHitbox)TriggerAreas[0]).Position = value;
                OnPositionChanged?.Invoke(this, new PositionEventArgs(value));
                OnChange?.Invoke(this, EventArgs.Empty);
            }
        }
        public float Width
        {
            get => Structure.Width;
            set
            {
                Structure.Width = value;
                BorderElement.Width = value;
                ((RectangleHitbox)TriggerAreas[0]).Width = value;
                OnResize?.Invoke(this, new ResizeEventArgs(Width, Height));
                OnChange?.Invoke(this, EventArgs.Empty);
            }
        }
        public float Height
```

```
{
    get => Structure.Height;
    set
    {
        Structure.Height = value;
        BorderElement.Height = value;
        ((RectangleHitbox)TriggerAreas[0]).Height = value;
        OnResize?.Invoke(this, new ResizeEventArgs(Width, Height));
        OnChange?.Invoke(this, EventArgs.Empty);
    }
}
public string Text
{
    get => Structure.Text;
    set
    {
        Structure.Text = value;
        TextElement.Text = value;
        OnTextChange?.Invoke(this, new TextEventArgs(value));
        OnChange?.Invoke(this, EventArgs.Empty);
    }
}

public Color BorderColor
{
    get => BorderStyle.BorderColor;
    set
    {
        BorderStyle.BorderColor = value;
        BorderElement.BorderColor = value;
        OnChange?.Invoke(this, EventArgs.Empty);
    }
}
public Color FillColor
{
    get => BorderStyle.FillColor;
    set
    {
        BorderStyle.FillColor = value;
        BorderElement.FillColor = value;
        OnChange?.Invoke(this, EventArgs.Empty);
    }
}
public int BorderWidth
{
    get => BorderStyle.BorderWidth;
    set
    {
        BorderStyle.BorderWidth = value;
        BorderElement.BorderWidth = value;
        OnChange?.Invoke(this, EventArgs.Empty);
    }
}
public Color TextColor
{
    get => TextStyle.Color;
    set
    {
        TextStyle.Color = value;
        TextElement.Color = value;
        OnChange?.Invoke(this, EventArgs.Empty);
    }
}
public FontStyle Style
{
    get => TextStyle.FontStyle;
    set
    {
        TextStyle.FontStyle = value;
        TextElement.FontStyle = value;
        OnChange?.Invoke(this, EventArgs.Empty);
    }
}

public EventHandler<TextEventArgs> OnTextChange { get; set; }

public Action ButtonAction { get; private set; }
```

```
        public EventHandler<ResizeEventArgs> OnResize { get; set; }
        public EventHandler OnChange { get; set; }
        public EventHandler<NodeEventArgs> OnRemoval { get; set; }
        public EventHandler<NodeEventArgs> OnFocused { get; set; }
        public EventHandler<NodeEventArgs> OnUnfocused { get; set; }
        public EventHandler OnMouseClick { get; set; }

        private RectangleRenderElement BorderElement;
        private TextRenderElement TextElement;

        public void Render(Renderer renderer)
        {
            BorderElement.Render(renderer);
            TextElement.Render(renderer);
        }
    }
}
```

ClassDiagramNode.cs:

```
using UML_Editor.Enums;
using UML_Editor.Rendering;
using UML_Editor.Rendering.RenderingElements;
using System.Drawing;
using UML_Editor.EventArguments;
using UML_Editor.Hitboxes;
using UML_Editor.Rendering.ElementStyles;
using UML_Editor.Geometry;
using UML_Editor.NodeStructure;
using UML_Editor.ProjectStructure;

namespace UML_Editor.Nodes
{
    public class ClassDiagramNode : BasicContainerNode, IOptionsNode, IFocusableNode
    {
        public ClassStructure CodeStructure { get; set; }
        public List<PropertyNode> Properties = new List<PropertyNode>();
        public List<MethodNode> Methods = new List<MethodNode>();
        public BasicContainerNode OptionsPrefab { get; set; }
        public BasicContainerNode OptionsMenu { get; set; }
        private TextBoxNode NameTextBox;
        private LineRenderElement NameLine;
        private LineRenderElement SeparatorLine;

        public ClassDiagramNode(ClassStructure codestructure, BasicNodeStructure structure, RectangleRenderElementStyle
border_style) : base(structure, border_style)
        {
            CodeStructure = codestructure;
            NameTextBox = new TextBoxNode(new BasicTextNodeStructure(Position, Renderer.GetTextWidth(Name.Length),
Renderer.SingleTextHeight, Name), TextRenderElementStyle.Default, RectangleRenderElementStyle.Textbox);
            NameLine = new LineRenderElement(new Vector(Position.X, Position.Y + Renderer.SingleTextHeight), new
Vector(Position.X + Width, Position.Y + Renderer.SingleTextHeight), 1, Color.Black);
            SeparatorLine = new LineRenderElement(new Vector(Position.X, Position.Y + Renderer.SingleTextHeight), new
Vector(Position.X + Width, Position.Y + Renderer.SingleTextHeight), 1, Color.Black);
            Children.Add(NameTextBox);
            GeneratePrefab();
            RepositionChildren();
            SetEvents();
        }
        public void UpdateStructure(object sender, EventArgs e)
        {
            CodeStructure.Name = NameTextBox.Text;
        }
        public void SetEvents()
        {
            OnOptionsShow += ShowOptions;
            OnOptionsHide += HideOptions;
            OnUnfocused += OnUnFocus;
            OnChange += HideOptions;
            OnChange += UpdateStructure;
            //OnChange += (sender, args) => OnUnFocus(this, new NodeEventArgs(this));
            Children.ForEach(x => x.OnResize += OnChildResize);
            Children.OfType<IFocusableNode>().ToList().ForEach(x =>
            {
                x.OnFocused += OnNodeFocus;
                x.OnFocused += HideOptions;
                x.OnUnfocused += OnNodeUnfocus;
                x.OnRemoval += (sender, args) => RemoveNode(args.Node);
            });
        }
```

```
public virtual string Name
{
    get => CodeStructure.Name;
    set
    {
        CodeStructure.Name = value;
        NameTextBox.Text = value;
        OnCodeStructureChange?.Invoke(this, new CodeStructureEventArgs(CodeStructure));
        OnChange?.Invoke(this, EventArgs.Empty);
    }
}

public override void AddNode(INode node)
{
    if (node is MethodNode mn)
    {
        Methods.Add(mn);
        mn.OnHitboxCreation += AddHitbox;
        mn.OnHitboxDeletion += RemoveHitbox;
        mn.OnUnfocused += OnNodeUnfocus;
        mn.OnFocused += OnNodeFocus;
        mn.OnRemoval += (sender, args) => RemoveNode(args.Node);
        CodeStructure.Methods.Add(mn.CodeStructure);
        Height += Renderer.SingleTextHeight;
        mn.RepositionChildren();
        mn.SetEvents();
        base.AddNode(node);
    }
    else if (node is PropertyNode pn)
    {
        Properties.Add(pn);
        pn.OnHitboxCreation += AddHitbox;
        pn.OnHitboxDeletion += RemoveHitbox;
        pn.OnUnfocused += OnNodeUnfocus;
        pn.OnFocused += OnNodeFocus;
        pn.OnRemoval += (sender, args) => RemoveNode(args.Node);
        CodeStructure.Properties.Add(pn.CodeStructure);
        Height += Renderer.SingleTextHeight;
        pn.RepositionChildren();
        pn.SetEvents();
        base.AddNode(node);
    }
}

public override void RemoveNode(INode node)
{
    if (node is MethodNode mn)
    {
        Methods.Remove(mn);
        CodeStructure.Methods.Remove(mn.CodeStructure);
    }
    else if (node is PropertyNode pn)
    {
        Properties.Remove(pn);
        CodeStructure.Properties.Remove(pn.CodeStructure);
    }
    Height -= Renderer.SingleTextHeight;
    base.RemoveNode(node);
}

public virtual AccessModifiers AccessModifier
{
    get => CodeStructure.AccessModifier;
    set
    {
        CodeStructure.AccessModifier = value;
        OnCodeStructureChange?.Invoke(this, new CodeStructureEventArgs(CodeStructure));
        OnChange?.Invoke(this, EventArgs.Empty);
    }
}
public virtual Modifiers Modifier
{
    get => CodeStructure.Modifier;
    set
    {
        CodeStructure.Modifier = value;
        OnCodeStructureChange?.Invoke(this, new CodeStructureEventArgs(CodeStructure));
        OnChange?.Invoke(this, EventArgs.Empty);
```

```
            }
        }
        public float GetWidest()
        {
            INode temp_node = Children.OrderByDescending(x => x.Width).FirstOrDefault();
            return temp_node.Width;
        }
        public override void RepositionChildren()
        {
            Width = GetWidest();
            NameTextBox.Position = new Vector((Position.X + Width / 2) - (NameTextBox.Width / 2), Position.Y);
            for (int i = 0; i < Properties.Count; i++)
            {
                Vector newpos = Position + new Vector(0, (i + 1) * Renderer.SingleTextHeight);
                if(Properties[i].Position != newpos)
                    Properties[i].Position = newpos;
            }
            for (int i = 0; i < Methods.Count; i++)
            {
                Vector newpos = Position + new Vector(0, (i + Properties.Count + 1) * Renderer.SingleTextHeight);
                if (Methods[i].Position != newpos)
                    Methods[i].Position = newpos;
            }
            NameLine.StartPoint = new Vector(Position.X, Position.Y + Renderer.SingleTextHeight);
            NameLine.EndPoint = new Vector(Position.X + Width, Position.Y + Renderer.SingleTextHeight);
            if (Properties.Count > 0)
            {
                PropertyNode prop = Properties.Last();
                SeparatorLine.StartPoint = prop.Position + new Vector(0, Renderer.SingleTextHeight);
                SeparatorLine.EndPoint = prop.Position + new Vector(Width, Renderer.SingleTextHeight);
            }
            else
            {
                SeparatorLine = new LineRenderElement(new Vector(Position.X, Position.Y + Renderer.SingleTextHeight), new
Vector(Position.X + Width, Position.Y + Renderer.SingleTextHeight), 1, Color.Black);
            }
        }
        public override void OnNodeFocus(object sender, NodeEventArgs e)
        {
            if (FocusedNode != e.Node)
            {
                OnFocused?.Invoke(this, new NodeEventArgs(this));
                FocusedNode?.OnUnfocused?.Invoke(this, new NodeEventArgs(FocusedNode));
                FocusedNode = (IFocusableNode)e.Node;
            }
        }
        public void OnUnFocus(object sender, NodeEventArgs e)
        {
            FocusedNode?.OnUnfocused?.Invoke(this, new NodeEventArgs(FocusedNode));
            OnOptionsHide?.Invoke(this, EventArgs.Empty);
        }
        public override void OnNodeUnfocus(object sender, NodeEventArgs e)
        {
            if (FocusedNode == e.Node)
                FocusedNode = null;
        }
        public void GeneratePrefab()
        {
            float total_Width = Renderer.GetTextWidth(13);
            OptionsPrefab = new BasicContainerNode(new BasicNodeStructure(Vector.Zero, total_Width, Renderer.SingleTextHeight *
3), RectangleRenderElementStyle.Default);
            OptionsPrefab.AddNode(new ButtonNode(new ButtonStructure(Vector.Zero, "Add Property", total_Width,
Renderer.SingleTextHeight, () =>
                {
                    AddNode(new PropertyNode(new PropertyStructure(Vector.Zero, "Property", "String", AccessModifiers.Public,
Modifiers.None), new BasicNodeStructure(Vector.Zero, 0, Renderer.SingleTextHeight),
                        RectangleRenderElementStyle.Textbox));
                    OnOptionsHide?.Invoke(this, EventArgs.Empty);
                }),
                RectangleRenderElementStyle.Default,
                TextRenderElementStyle.Default));
            OptionsPrefab.AddNode(new ButtonNode(new ButtonStructure(Vector.Zero, "AddMethod", total_Width,
Renderer.SingleTextHeight, () =>
                {
                    AddNode(new MethodNode(new MethodStructure(Vector.Zero, "Method", "void", "", AccessModifiers.Public,
Modifiers.None), new BasicNodeStructure(Vector.Zero, 0, Renderer.SingleTextHeight),
                        RectangleRenderElementStyle.Textbox));
                    OnOptionsHide?.Invoke(this, EventArgs.Empty);
```

```
                    }),
                    RectangleRenderElementStyle.Default,
                    TextRenderElementStyle.Default));
                OptionsPrefab.AddNode(new ButtonNode(new ButtonStructure(Vector.Zero, "Remove", total_Width,
Renderer.SingleTextHeight, () =>
                {
                    OnRemoval?.Invoke(this, new NodeEventArgs(this));
                    OnOptionsHide?.Invoke(this, EventArgs.Empty);
                    }),
                    RectangleRenderElementStyle.Default,
                    TextRenderElementStyle.Default));
            }
            public EventHandler OnOptionsShow { get; set; }
            public EventHandler OnOptionsHide { get; set; }
            public EventHandler<NodeEventArgs> OnFocused { get; set; }
            public EventHandler<NodeEventArgs> OnUnfocused { get; set; }
            public EventHandler OnMouseClick { get; set; }
            public EventHandler<CodeStructureEventArgs> OnCodeStructureChange { get; set; }
            public EventHandler<HitboxEventArgs> OnHitboxCreation { get; set; }
            public EventHandler<HitboxEventArgs> OnHitboxDeletion { get; set; }
            public void AddHitbox(object sender, HitboxEventArgs e)
            {
                TriggerAreas.Add(e.Hitbox);
            }

            public void RemoveHitbox(object sender, HitboxEventArgs e)
            {
                TriggerAreas.Remove(e.Hitbox);
            }
            public void AddHitbox(IHitbox hitbox)
            {
                TriggerAreas.Add(hitbox);
                OnHitboxCreation?.Invoke(this, new HitboxEventArgs(hitbox));
            }

            public void RemoveHitbox(IHitbox hitbox)
            {
                TriggerAreas.Remove(hitbox);
                OnHitboxDeletion?.Invoke(this, new HitboxEventArgs(hitbox));
            }
            public void ShowOptions(object sender, EventArgs e)
            {
                if (OptionsMenu == null)
                {
                    OptionsMenu = OptionsPrefab;
                    PrependNode(OptionsMenu);
                    OnFocused?.Invoke(this, new NodeEventArgs(this));
                    FocusedNode?.OnUnfocused?.Invoke(this, new NodeEventArgs(FocusedNode));
                    AddHitbox(OptionsMenu.TriggerAreas[0]);
                }
                else
                    OnOptionsHide?.Invoke(this, e);
            }
            public void HideOptions(object sender, EventArgs e)
            {
                if (OptionsMenu != null)
                {
                    RemoveHitbox(OptionsMenu.TriggerAreas[0]);
                    Children.Remove(OptionsMenu);
                }
                OptionsMenu = null;
            }

            public override void Render(Renderer renderer)
            {
                base.Render(renderer);
                FocusedNode?.Render(renderer);
                SeparatorLine.Render(renderer);
                NameLine.Render(renderer);
                if (FocusedNode != null && FocusedNode is PropertyNode pn)
                {
                    pn.OptionsMenu?.Render(renderer);
                    pn.AccessModifierMenu?.Render(renderer);
                }
                BorderElement.BorderOnly(renderer);
                OptionsMenu?.Render(renderer);
            }
```

```
        public bool IsOnEdge(Vector v)
        {
            float left = Position.X;
            float right = Position.X + Width;
            float top = Position.Y;
            float bot = Position.Y + Height;
            return v.X == left || v.X == right || v.Y == top || v.Y == bot;
        }

        public List<Vector> GetSideCenters()
        {
            float left = Position.X;
            float right = Position.X + Width;
            float top = Position.Y;
            float bot = Position.Y + Height;
            List<Vector> centers = new List<Vector>();
            centers.Add(new Vector((left + right) / 2, top));
            centers.Add(new Vector(right, (top + bot) / 2));
            centers.Add(new Vector((left + right) / 2, bot));
            centers.Add(new Vector(left, (top + bot) / 2));
            return centers;
        }

        public Vector GetTopAnchor() => new Vector((Position.X + (Width / 2)), Position.Y);
        public Vector GetBotAnchor() => new Vector((Position.X + (Width / 2)), Position.Y + Height);
        public Vector GetLeftAnchor() => new Vector(Position.X, Position.Y + (Height / 2));
        public Vector GetRightAnchor() => new Vector(Position.X + Width, Position.Y + (Height / 2));
        public Vector GetCenter() => new Vector(Position.X + (Width / 2), Position.Y + (Height / 2));
        public Vector GetTopLeftCorner() => new Vector(Position.X, Position.Y);
        public Vector GetTopRightCorner() => new Vector(Position.X + Width, Position.Y);
        public Vector GetBotLeftCorner() => new Vector(Position.X, Position.Y + Height);
        public Vector GetBotRightCorner() => new Vector(Position.X + Width, Position.Y + Height);
        public Line GetTopSide() => new Line(GetTopLeftCorner(), GetTopRightCorner());
        public Line GetBotSide() => new Line(GetBotLeftCorner(), GetBotRightCorner());
        public Line GetLeftSide() => new Line(GetTopLeftCorner(), GetBotLeftCorner());
        public Line GetRightSide() => new Line(GetTopRightCorner(), GetBotRightCorner());

    }
}
```

LabelNode.cs:

```
using UML_Editor.EventArguments;
using UML_Editor.Geometry;
using UML_Editor.Hitboxes;
using UML_Editor.NodeStructure;
using UML_Editor.Rendering;
using UML_Editor.Rendering.ElementStyles;
using UML_Editor.Rendering.RenderingElements;

namespace UML_Editor.Nodes
{
    public class LabelNode : BasicNode, ITextNode
    {
        public new BasicTextNodeStructure Structure { get; set; }
        public TextRenderElementStyle TextStyle { get; }

        private TextRenderElement TextElement;

        public LabelNode(BasicTextNodeStructure structure, TextRenderElementStyle text_style,
            RectangleRenderElementStyle border_style) : base(structure, border_style)
        {
            Structure = structure;
            TextStyle = text_style;
            TextElement = new TextRenderElement(Position, Text, text_style.Color, text_style.FontSize, text_style.FontStyle);
        }
        public string Text
        {
            get => Structure.Text;
            set
            {
                Structure.Text = value;
                TextElement.Text = value;
                Width = Renderer.GetTextWidth(value.Length);
                OnTextChange?.Invoke(this, new TextEventArgs(Text));
                OnChange?.Invoke(this, EventArgs.Empty);
            }
        }
```

```
        public override Vector Position
        {
            get => base.Position;
            set
            {
                TextElement.Position = value;
                base.Position = value;
            }
        }

        public Color TextColor
        {
            get => TextStyle.Color;
            set
            {
                TextStyle.Color = value;
                TextElement.Color = value;
                OnChange?.Invoke(this, EventArgs.Empty);
            }
        }
        public FontStyle Style
        {
            get => TextStyle.FontStyle;
            set
            {
                TextStyle.FontStyle = value;
                TextElement.FontStyle = value;
                OnChange?.Invoke(this, EventArgs.Empty);
            }
        }

        public override void Render(Renderer renderer)
        {
            base.Render(renderer);
            TextElement.Render(renderer);
        }
        public EventHandler<TextEventArgs> OnTextChange { get; set; }
    }
}
```

MethodNode.cs:

```
using UML_Editor.Enums;
using UML_Editor.Geometry;
using UML_Editor.Hitboxes;
using UML_Editor.NodeStructure;
using UML_Editor.Rendering;
using UML_Editor.Rendering.ElementStyles;
using UML_Editor.Rendering.RenderingElements;
using UML_Editor.ProjectStructure;

namespace UML_Editor.Nodes
{
    public class MethodNode : PropertyNode
    {
        public new MethodStructure CodeStructure { get; set; }

        public TextBoxNode ArgumentsTextBox { get; set; }
        public LabelNode LeftBracket { get; set; }
        public LabelNode RightBracket { get; set; }

        public MethodNode(MethodStructure codeStructure, BasicNodeStructure structure, RectangleRenderElementStyle border_style)
            : base(new PropertyStructure(codeStructure.Position, codeStructure.Name, codeStructure.Type,
codeStructure.AccessModifier, codeStructure.Modifier), structure, border_style)
        {
            CodeStructure = codeStructure;
            LeftBracket = new LabelNode(new BasicTextNodeStructure(Position + new Vector(AccessModifierButton.Width +
NameTextBox.Width, 0),
                Renderer.SingleTextWidth, Height, "("),
                TextRenderElementStyle.Default, RectangleRenderElementStyle.Textbox);
            string argum = codeStructure.Arguments;
            ArgumentsTextBox = new TextBoxNode(new BasicTextNodeStructure(Position + new Vector(AccessModifierButton.Width
+ NameTextBox.Width + LeftBracket.Width, 0),
                Renderer.GetTextWidth(argum.Length), Height, argum),
                TextRenderElementStyle.Default, RectangleRenderElementStyle.Textbox);
            RightBracket = new LabelNode(new BasicTextNodeStructure(Position + new Vector(NameTextBox.Width +
AccessModifierButton.Width + ArgumentsTextBox.Width + LeftBracket.Width, 0),
                    Renderer.SingleTextWidth, Height, ")"),
                TextRenderElementStyle.Default, RectangleRenderElementStyle.Textbox);
            Children.Add(LeftBracket);
```

```
            Children.Add(RightBracket);
            Children.Add(ArgumentsTextBox);
            GenerateMenu();
            GenerateOptions();
        }

        public override void UpdateStructure(object sender, EventArgs e)
        {
            CodeStructure.Name = NameTextBox.Text;
            CodeStructure.Type = TypeTextBox.Text;
            CodeStructure.Arguments = ArgumentsTextBox.Text;
        }

        public override float GetWidth()
        {
            return AccessModifierButton.Width + NameTextBox.Width + Separator.Width + TypeTextBox.Width + LeftBracket.Width +
RightBracket.Width + ArgumentsTextBox.Width;
        }

        public override void RepositionChildren()
        {
            Width = GetWidth();
            AccessModifierButton.Position = new Vector(Position.X, Position.Y);
            NameTextBox.Position = Position + new Vector(AccessModifierButton.Width, 0);
            ArgumentsTextBox.Position = Position + new Vector(AccessModifierButton.Width + NameTextBox.Width +
LeftBracket.Width, 0);
            TypeTextBox.Position = new Vector(Position.X + AccessModifierButton.Width + NameTextBox.Width +
ArgumentsTextBox.Width + LeftBracket.Width + RightBracket.Width + Separator.Width, Position.Y);
            LeftBracket.Position = Position + new Vector(NameTextBox.Width + AccessModifierButton.Width, 0);
            RightBracket.Position = Position + new Vector(NameTextBox.Width + AccessModifierButton.Width +
ArgumentsTextBox.Width + LeftBracket.Width, 0);
            Separator.Position = Position + new Vector(NameTextBox.Width + AccessModifierButton.Width +
ArgumentsTextBox.Width + LeftBracket.Width + RightBracket.Width, 0);
        }
    }
}
```

PropertyNode.cs:

```
using UML_Editor.Rendering;
using UML_Editor.Enums;
using UML_Editor.Rendering.RenderingElements;
using UML_Editor.Rendering.ElementStyles;
using UML_Editor.Hitboxes;
using System.Drawing;
using System.Windows.Forms;
using UML_Editor.EventArguments;
using UML_Editor.Geometry;
using UML_Editor.NodeStructure;
using UML_Editor.ProjectStructure;

namespace UML_Editor.Nodes
{
    public class PropertyNode : BasicContainerNode, IOptionsNode
    {
        public BasicContainerNode AccessModifierMenu { get; set; }
        public BasicContainerNode MenuPrefab { get; set; }
        public ButtonNode AccessModifierButton { get; set; }
        public TextBoxNode TypeTextBox { get; set; }
        public TextBoxNode NameTextBox { get; set; }
        public LabelNode Separator { get; set; }

        public PropertyStructure CodeStructure { get; set; }

        public PropertyNode(PropertyStructure codeStructure, BasicNodeStructure structure, RectangleRenderElementStyle
border_style) : base(structure, border_style)
        {
            CodeStructure = codeStructure;
            AccessModifierButton = new ButtonNode(new ButtonStructure(Position, "+", Renderer.SingleTextWidth, Height, () =>
OnMenuShow?.Invoke(this, EventArgs.Empty)), RectangleRenderElementStyle.Textbox, TextRenderElementStyle.Default);
            NameTextBox = new TextBoxNode( new BasicTextNodeStructure(Position, Renderer.GetTextWidth(Name.Length), Height,
Name), TextRenderElementStyle.Default, RectangleRenderElementStyle.Textbox);
            Separator = new LabelNode(new BasicTextNodeStructure(Position, Renderer.SingleTextWidth, Height, ":"),
TextRenderElementStyle.Default, RectangleRenderElementStyle.Textbox);
            TypeTextBox = new TextBoxNode(new BasicTextNodeStructure(Position, Renderer.GetTextWidth(Type.Length), Height,
Type), TextRenderElementStyle.Default, RectangleRenderElementStyle.Textbox);
            Children.Add(AccessModifierButton);
            Children.Add(NameTextBox);
            Children.Add(Separator);
            Children.Add(TypeTextBox);
```

```
    OnUnfocused += OnUnFocus;
    GenerateMenu();
    GenerateOptions();
}

public virtual void UpdateStructure(object sender, EventArgs e)
{
    CodeStructure.Name = NameTextBox.Text;
    CodeStructure.Type = TypeTextBox.Text;
}

public void SetEvents()
{
    OnMenuHide += HideMenu;
    OnMenuShow += ShowMenu;
    OnOptionsShow += ShowOptions;
    OnOptionsHide += HideOptions;
    OnOptionsShow += HideMenu;
    OnMenuShow += HideOptions;
    OnChange += HideMenu;
    OnChange += HideOptions;
    OnChange += UpdateStructure;
    Children.ForEach(x => x.OnResize += OnChildResize);
    Children.OfType<IFocusableNode>().ToList().ForEach(x =>
    {
        if(!(x is ButtonNode))
        {
            x.OnFocused += HideMenu;
            x.OnFocused += HideOptions;
            x.OnFocused += OnNodeFocus;
        }
        x.OnUnfocused += OnNodeUnfocus;
    });
}
public virtual string Name
{
    get => CodeStructure.Name;
    set
    {
        CodeStructure.Name = value;
        NameTextBox.Text = value;
        OnCodeStructureChange?.Invoke(this, new CodeStructureEventArgs(CodeStructure));
        OnChange?.Invoke(this, EventArgs.Empty);
    }
}
public virtual string Type
{
    get => CodeStructure.Type;
    set
    {
        CodeStructure.Type = value;
        TypeTextBox.Text = value;
        OnCodeStructureChange?.Invoke(this, new CodeStructureEventArgs(CodeStructure));
        OnChange?.Invoke(this, EventArgs.Empty);
    }
}

public virtual AccessModifiers AccessModifier
{
    get => CodeStructure.AccessModifier;
    set
    {
        CodeStructure.AccessModifier = value;
        SetAccessModifier();
        OnCodeStructureChange?.Invoke(this, new CodeStructureEventArgs(CodeStructure));
        OnChange?.Invoke(this, EventArgs.Empty);
    }
}
public virtual Modifiers Modifier
{
    get => CodeStructure.Modifier;
    set
    {
        CodeStructure.Modifier = value;
        SetModifier();
        OnCodeStructureChange?.Invoke(this, new CodeStructureEventArgs(CodeStructure));
        OnChange?.Invoke(this, EventArgs.Empty);
    }
```

```
        }

        public virtual float GetWidth() => AccessModifierButton.Width + NameTextBox.Width + Separator.Width +
TypeTextBox.Width;

        public void OnUnFocus(object sender, NodeEventArgs e)
        {
            FocusedNode?.OnUnfocused?.Invoke(this, new NodeEventArgs(FocusedNode));
            OnMenuHide?.Invoke(this, EventArgs.Empty);
            OnOptionsHide?.Invoke(this, EventArgs.Empty);
        }
        public override void OnNodeFocus(object sender, NodeEventArgs e)
        {
            if (FocusedNode != e.Node && !(e.Node is ButtonNode))
            {
                OnFocused?.Invoke(this, new NodeEventArgs(this));
                FocusedNode?.OnUnfocused?.Invoke(this, new NodeEventArgs(FocusedNode));
                FocusedNode = (IFocusableNode)e.Node;
            }
        }
        public override void OnNodeUnfocus(object sender, NodeEventArgs e)
        {
            if (FocusedNode == e.Node)
                FocusedNode = null;
        }
        public override void RepositionChildren()
        {
            Width = GetWidth();
            if (Height != Renderer.SingleTextHeight)
                Height = Renderer.SingleTextHeight;
            AccessModifierButton.Position = new Vector(Position.X, Position.Y);
            NameTextBox.Position = new Vector(Position.X + AccessModifierButton.Width, Position.Y);
            Separator.Position = new Vector(Position.X + AccessModifierButton.Width + NameTextBox.Width, Position.Y);
            TypeTextBox.Position = new Vector(Position.X + AccessModifierButton.Width + NameTextBox.Width + Separator.Width,
Position.Y);
        }

        public override void Render(Renderer renderer)
        {
            base.Render(renderer);
            OptionsMenu?.Render(renderer);
            AccessModifierMenu?.Render(renderer);
        }

        public virtual void SetAccessModifier()
        {
            switch (AccessModifier)
            {
                case AccessModifiers.Private:
                    AccessModifierButton.Text = "-";
                    break;
                case AccessModifiers.Public:
                    AccessModifierButton.Text = "+";
                    break;
                case AccessModifiers.Protected:
                    AccessModifierButton.Text = "#";
                    break;
                default:
                    AccessModifierButton.Text = "E";
                    break;
            }
        }

        public virtual void SetModifier()
        {
            switch (Modifier)
            {
                case Modifiers.None:
                    NameTextBox.Style = FontStyle.Regular;
                    break;
                case Modifiers.Static:
                    NameTextBox.Style = FontStyle.Underline;
                    break;
                case Modifiers.Abstract:
                    NameTextBox.Style = FontStyle.Italic;
                    break;
            }
        }
```

```
        public EventHandler<NodeEventArgs> OnFocused { get; set; }
        public EventHandler<NodeEventArgs> OnUnfocused { get; set; }
        public EventHandler<CodeStructureEventArgs> OnCodeStructureChange { get; set; }
        public EventHandler OnMouseClick { get; set; }
        public BasicContainerNode OptionsPrefab { get; set; }
        public BasicContainerNode OptionsMenu { get; set; }
        public virtual void GenerateOptions()
        {
            float total_Width = Renderer.GetTextWidth(13);
            OptionsPrefab = new BasicContainerNode(new BasicNodeStructure(Vector.Zero, total_Width, Renderer.SingleTextHeight *
4), RectangleRenderElementStyle.Default);
            OptionsPrefab.AddNode(new ButtonNode(new ButtonStructure(Vector.Zero, "Make Regular", total_Width,
Renderer.SingleTextHeight , () =>
                {
                    Modifier = Modifiers.None;
                    OnOptionsHide?.Invoke(this, EventArgs.Empty);
                }),
                RectangleRenderElementStyle.Default,
                TextRenderElementStyle.Default));
            OptionsPrefab.AddNode(new ButtonNode(new ButtonStructure(Vector.Zero, "Make Abstract", total_Width,
Renderer.SingleTextHeight, () =>
                {
                    Modifier = Modifiers.Abstract;
                    OnOptionsHide?.Invoke(this, EventArgs.Empty);
                }),
                RectangleRenderElementStyle.Default,
                TextRenderElementStyle.Default));
            OptionsPrefab.AddNode(new ButtonNode(new ButtonStructure(Vector.Zero, "Make Static", total_Width,
Renderer.SingleTextHeight, () =>
                {
                    Modifier = Modifiers.Static;
                    OnOptionsHide?.Invoke(this, EventArgs.Empty);
                }),
                RectangleRenderElementStyle.Default,
                TextRenderElementStyle.Default));
            OptionsPrefab.AddNode(new ButtonNode(new ButtonStructure(Vector.Zero, "Remove", total_Width,
Renderer.SingleTextHeight, () =>
                {
                    OnRemoval?.Invoke(this, new NodeEventArgs(this));
                    OnOptionsHide?.Invoke(this, EventArgs.Empty);
                }),
                RectangleRenderElementStyle.Default,
                TextRenderElementStyle.Default));
        }

        public virtual void GenerateMenu()
        {
            float total_Width = Renderer.GetTextWidth(9);
            MenuPrefab = new BasicContainerNode(new BasicNodeStructure(Vector.Zero, total_Width, Renderer.SingleTextHeight * 3),
RectangleRenderElementStyle.Default);
            MenuPrefab.AddNode(new ButtonNode(new ButtonStructure(Vector.Zero, "Public", total_Width, Renderer.SingleTextHeight,
() =>
                {
                    AccessModifier = AccessModifiers.Public;
                    OnMenuHide?.Invoke(this, EventArgs.Empty);
                }),
                RectangleRenderElementStyle.Default,
                TextRenderElementStyle.Default));
            MenuPrefab.AddNode(new ButtonNode(new ButtonStructure(Vector.Zero, "Private", total_Width,
Renderer.SingleTextHeight, () =>
                {
                    AccessModifier = AccessModifiers.Private;
                    OnMenuHide?.Invoke(this, EventArgs.Empty);
                }),
                RectangleRenderElementStyle.Default,
                TextRenderElementStyle.Default));
            MenuPrefab.AddNode(new ButtonNode(new ButtonStructure(Vector.Zero, "Protected", total_Width,
Renderer.SingleTextHeight, () =>
                {
                    AccessModifier = AccessModifiers.Protected;
                    OnMenuHide?.Invoke(this, EventArgs.Empty);
                }),
                RectangleRenderElementStyle.Default,
                TextRenderElementStyle.Default));
        }

        public void AddHitbox(IHitbox hitbox)
```

```
        {
            TriggerAreas.Add(hitbox);
            OnHitboxCreation?.Invoke(this, new HitboxEventArgs(hitbox));
        }

        public void RemoveHitbox(IHitbox hitbox)
        {
            TriggerAreas.Remove(hitbox);
            OnHitboxDeletion?.Invoke(this, new HitboxEventArgs(hitbox));
        }
        public void ShowMenu(object sender, EventArgs e)
        {
            if (AccessModifierMenu == null)
            {
                MenuPrefab.Position = AccessModifierButton.Position + new Vector(AccessModifierButton.Width, 0);
                AccessModifierMenu = MenuPrefab;
                PrependNode(AccessModifierMenu);
                OnFocused?.Invoke(this, new NodeEventArgs(this));
                AddHitbox(AccessModifierMenu.TriggerAreas[0]);
            }
        }
        public void HideMenu(object sender, EventArgs e)
        {
            if (AccessModifierMenu != null)
            {
                RemoveHitbox(AccessModifierMenu.TriggerAreas[0]);
                Children.Remove(AccessModifierMenu);
            }
            AccessModifierMenu = null;
        }
        public void ShowOptions(object sender, EventArgs e)
        {
            if (OptionsMenu == null)
            {
                OptionsMenu = OptionsPrefab;
                PrependNode(OptionsMenu);
                OnFocused?.Invoke(this, new NodeEventArgs(this));
                AddHitbox(OptionsMenu.TriggerAreas[0]);
            }
            else
                OnOptionsHide?.Invoke(this, e);
        }
        public void HideOptions(object sender, EventArgs e)
        {
            if(OptionsMenu != null)
            {
                RemoveHitbox(OptionsMenu.TriggerAreas[0]);
                Children.Remove(OptionsMenu);
            }
            OptionsMenu = null;
        }
        public EventHandler OnOptionsShow { get; set; }
        public EventHandler OnOptionsHide { get; set; }
        public EventHandler OnMenuShow { get; set; }
        public EventHandler OnMenuHide { get; set; }
        public EventHandler<HitboxEventArgs> OnHitboxCreation { get; set; }
        public EventHandler<HitboxEventArgs> OnHitboxDeletion { get; set; }
    }
}
```

TextBoxNode.cs:

```
using UML_Editor.EventArguments;
using UML_Editor.Hitboxes;
using UML_Editor.Geometry;
using UML_Editor.NodeStructure;
using UML_Editor.Rendering.ElementStyles;

namespace UML_Editor.Nodes
{
    public class TextBoxNode : LabelNode, IKeyboardFocusableNode
    {
        public TextBoxNode(BasicTextNodeStructure structure, TextRenderElementStyle text_style,
            RectangleRenderElementStyle border_style) : base(structure, text_style, border_style)
        {
            TriggerAreas.Add(new RectangleHitbox(Position, Width, Height));
            OnKeyPress += HandleKey;
            OnFocused += (sender, args) => FillColor = Color.CadetBlue;
            OnUnfocused += (sender, args) => FillColor = Color.White;
        }
```

```
        private void HandleKey(object sender, KeyPressEventArgs e)
        {
            char key = e.KeyChar;
            if (key == (char)8 && Text.Length > 0)
            {
                Text = Text.Substring(0, Text.Length - 1);
                Width = Renderer.GetTextWidth(Text.Length);
            }
            else if (key == (char)13)
                OnUnfocused?.Invoke(this, new NodeEventArgs(this));
            else if (Char.IsWhiteSpace(key))
                Text = Text.Insert(Text.Length, " ");
            else
                Text = Text.Insert(Text.Length, key.ToString());
        }

        public EventHandler<KeyPressEventArgs> OnKeyPress { get; set; }
        public EventHandler<NodeEventArgs> OnFocused { get; set; }
        public EventHandler<NodeEventArgs> OnUnfocused { get; set; }
    }
}
```

BasicNodeStructure.cs:

```
using UML_Editor.Geometry;

namespace UML_Editor.NodeStructure
{
    public class BasicNodeStructure
    {
        public Vector Position { get; set; }
        public float Width { get; set; }
        public float Height { get; set; }

        public BasicNodeStructure(Vector pos, float width, float height)
        {
            Position = pos;
            Width = width;
            Height = height;
        }
    }
}
```

BasicTextNodeStructure.cs:

```
using UML_Editor.Geometry;

namespace UML_Editor.NodeStructure
{
    public class BasicTextNodeStructure : BasicNodeStructure
    {
        public BasicTextNodeStructure(Vector pos, float width, float height, string text) : base(pos, width, height)
        {
            Text = text;
        }

        public string Text { get; set; }
    }
}
```

ButtonStructure.cs:

```
using UML_Editor.Geometry;

namespace UML_Editor.NodeStructure
{
    public class ButtonStructure : BasicTextNodeStructure
    {
        public Action ButtonAction { get; set; }
        public string ButtonText { get; set; }
        public ButtonStructure(Vector pos, string text, float width, float height, Action action) : base(pos, width, height, text)
        {
            ButtonAction = action;
            ButtonText = text;
        }
    }
}
```