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 програмного забезпечення для інтелектуального текстового редактора»
за освітньою програмою: «Інженерія програмного забезпечення»
зі спеціальності: «121 Інженерія програмного забезпечення»
Виконав: студент групи «ПЗ1811»

_____ /Маріанна ІГНАШКІНА/
(підпис студента) (Ім'я ПРІЗВИЩЕ)

Керівник: _____ /ст. викл. Сергій САМОЙЛОВ/
(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: «Software development for intellectual text editor»
according to educational curriculum «Software engineering»
in the Speciality: «121 Software engineering»

Done by the student of the group PZ1811: /Marianna IHNASHKINA/

Scientific Supervisor: /Serhii SAMOILOV/

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 _____

З А В Д А Н Н Я

на кваліфікаційну роботу бакалавра
студенту Ігнашкіній Маріанні Валеріївні
1. Тема роботи: «Розробка програмного забезпечення для інтелектуального текстового редактора»
Керівник роботи: Самойлов Сергій Петрович, старший викладач
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 та аналіз вимог
Проєктування
Розробка програми
Тестування та налагодження
Аналіз та висновки
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень):
Презентація

КАЛЕНДАРНИЙ ПЛАН

№ п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
	Постановка задачі, збір інформації, виріб та обґрунтування критеріїв розробки	31.01.22 -31.02.22	
	Програмування та відлагодження програми.	01.03.22 - 15.05.22	
	Тестування програми	15.05.22 - 30.05.22	
	Розробка, узгодження і затвердження програмної документації.	31.05.22 - 12.06.22	
	Подання кваліфікаційної роботи до кафедри	07.06.22	
	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії	21.06.22	

Студент _____ Маріанна ІГНАШКІНА
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник роботи _____ ст. викл. Сергій САМОЙЛОВ
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка складається з 4 розділів:

– вступ – в даному розділі описується мета розробки, її актуальність та призначення. Складається з 2 сторінок;

– збір та аналіз вимог – в даному розділі розглядаються та описуються аналоги розроблюваного додатку. Описується функціонал аналогів, а також їх переваги та недоліки. Складається з 16 сторінок;

– проєктування – в даному розділі проводиться проєктування інтерфейсу користувача, опис взаємодії користувача з системою, проєктування динаміки системи, а також описується загальна схема роботи програми та визначаються складові системи (програми). Складається з 11 сторінок;

– розробка програми – в даному розділі розробляється основна логіка роботи програми, а також ствюються алгоритми, необхідні для роботи програми та окремих її складових. Складається з 10 сторінок;

– тестування та налагодження – в даному розділі описується вибір стратегії тестування, а також тестування методами «чорного» та «білого» ящика. Складається з 8 сторінок;

– висновки. Складається з 1 сторінки;

– список літератури – включає в себе бібліографічний список використаної літератури. Складається з 2 сторінок;

– додатки – містять вимоги до програмного застосунку і текст програми;

– кількість таблиць: 6 штук. Кількість рисунків: 17 штук;

– ключові слова: програмне забезпечення, інтелектуальний текстовий редактор, мови програмування, версії документу, плагін.

ЗМІСТ

ВСТУП

Об’єктом розробки даної дипломної роботи є програмне забезпечення для інтелектуального текстового редактора. На сьогоднішній день на ринку програмних засобів представлена велика кількість редакторів коду. Це програмні продукти з різним інтерфейсом користувача і обмеженим набором функцій для кожного з них. Однією з переваг, що зустрічається у багатьох платформ, є відкритий вихідний код, що дозволяє значно розширити їх функціональні можливості за допомогою додаткових плагінів або додаткового програмного забезпечення.

Для створення елементів ПЗ, а також окремих додатків використовуються інтелектуальні текстові редактори. Саме з їх допомогою робота над майбутніми програмами є комфортною та набуває високих показників продуктивності. Це робить тему дипломної роботи актуальною та затребуваною.

З розвитком інтелектуальних текстових редакторів також набувають актуальності плагіни та надбудови для них, тому розробка додаткових функцій, полегшуючих користування редакторами та розширюючих кордони його використання являється необхідними та затребуваними.

Мета роботи – розробка додаткового програмного забезпечення до існуючого інтелектуального текстового редактора задля адаптації до потреб або переваг користувачів та полегшення роботи з цим редактором.

Функціональне призначення – програмний продукт використовує різні версії документів користувача, або різних користувачів та полегшує формування нового документу на основі попередніх.

Експлуатаційне призначення – за допомогою розширення виконується зіставлення різних версій документів задля швидкого пошуку відмінностей та створення нового документу за бажанням користувача.

Результати роботи можуть стати основою для створення повномасштабного плагіну або надбудови для будь-якого інтелектуального текстового редактора з відкритим кодом та подальшого його впровадження.

РОЗДІЛ 1. ЗБІР ТА АНАЛІЗ ВИМОГ

Збір вимог – це один з найважливіших етапів процесу створення будь-якої інформаційної системи, будь то десктопне, веб або мобільний додаток або просто доопрацювання вже існуючого рішення. Перш ніж почати збирати вимоги, необхідно виявити всіх зацікавлених осіб, які користуватимуться системою. Чим точнішим буде цей список, тим повнішими будуть вимоги [1].

1.1 Методи збору вимог до програмного забезпечення

Існує безліч різних технік збору вимог, які допоможуть краще зрозуміти, що хоче замовник.

Розглянемо основні їх трохи докладніше.

Анкетування.

Даний спосіб має на увазі під собою складання листа-опитувальника (анкети, брифа), який може містити відкриті (вимагають від опитуваного сформулювати його відповідь) та закриті (вимагають від опитуваного вибрати відповідь із запропонованих варіантів) питання.

Анкетування використовується для того, щоб підтвердити або деталізувати відомі раніше вимоги, вибрати параметри для рішень.

Найвідомішим прикладом анкетування може бути "Бриф на розробку сайту" – анкета, яка містить список основних вимог та інформацію про майбутній сайт.

Переваги:

– висока швидкість отримання результатів;

– порівняно невеликі матеріальні витрати.

Недоліки:

– цей метод не підходить для виявлення неявних вимог;

– при складанні опитувальника фізично неможливо врахувати всі необхідні питання.

Інтерв'ю

Цей метод відомий багатьом, свого роду розмова "за душею" із зацікавленою особою, тет-а-тет.

Необхідно ставити відкриті питання для отримання інформації та закриті для того, щоб підтвердити або спростувати конкретні варіанти вимог.

Цей спосіб застосовується в основному для отримання інформації з будь-якої конкретної теми та/або для уточнення вимог.

Багатьом може здатись цей спосіб досить легким, але це не так. Провести гарне інтерв'ю досить складно. Ви повинні гнучко реагувати на реакцію інтерв'юйованого і в разі потреби змінювати порядок заготовлених питань або їх формулювання. Не забудьте увімкнути диктофон під час інтерв'ю або вести нотатки.

Переваги:

– можливість ставити запитання у довільній послідовності;

– можливість використання допоміжного матеріалу;

– аналіз невербальної реакції опитуваної людини дозволить зробити додатковий висновок про достовірність її відповідей.

Недоліки:

– інтерв'ю забирає досить багато часу та сил;

– додатковою складністю є отримання однакових відповідей від тих, хто інтерв'ює;

– вивчення існуючої документації.

Ця методика може бути використана за наявності в організації документації, яка може допомогти у визначенні потреб Замовника. Приклади документації включають: регламенти, описи процесів, структура організації, специфікації продукту, різні процедури, стандарти та інструкції, шаблони документів і т.д.

Виявлені вимоги є основою для подальшого аналізу та мають бути деталізовані.

Дана методика застосовна, наприклад, при автоматизації усталених в організації регламентованих бізнес-процесів.

Перевага – швидке отримання інформації.

Недолік – цей спосіб не застосовується за наявності в компанії лише базових документів (або за їх повної відсутності) або, якщо у компанії Замовника не підтримується актуальність документації.

Робота «у полі» складається зі спостереження за діяльністю та процесами майбутніх користувачів системи, та у визначенні вимог, що ґрунтуються на цьому спостереженні. Якщо говорити простіше, то це спостереження за тим, як працюють користувачі, та документування процесу, завдань та результатів їх діяльності.

Метод дозволяє уникнути проблем, пов'язаних із труднощами користувачів в описі та висловленні своїх потреб. У деяких випадках процес спостереження може супроводжуватися інтерв'юванням користувачів для уточнення особливостей і деталей їх роботи і завдань. У процесі спостереження можна виявити шляхи оптимізації бізнес процесів Замовника.

Переваги:

– дозволяє наочно побачити проблему та розробити найоптимальніший варіант її вирішення;

– допомагає найточніше зібрати вимоги, спостерігаючи за роботою працівників.

Недоліки:

– у процесі спостереження можуть бути втрачені деякі альтернативні сценарії бізнес-процесу;

– важко застосуємо на секретних підприємствах чи небезпечних (шкідливих) виробництвах.

1.2 Збір вимог до програмного забезпечення, що розроблюється

Для цілей проєкту було обрано анкетування серед цільової аудиторії користувачів, а саме студентів ІТ-спеціальностей та фахівців цієї сфери.

В ході анкетування було визначено, що найпопулярніший текстовий редактор серед фахівців – Visual Studio Code. На рисунку 1.1 нижче можемо бачити статистику популярності текстових редакторів серед анкетованих фахівців, найбільш вживаним є редактор Visual Studio Code, в якому працюють понад 60% опитаних.

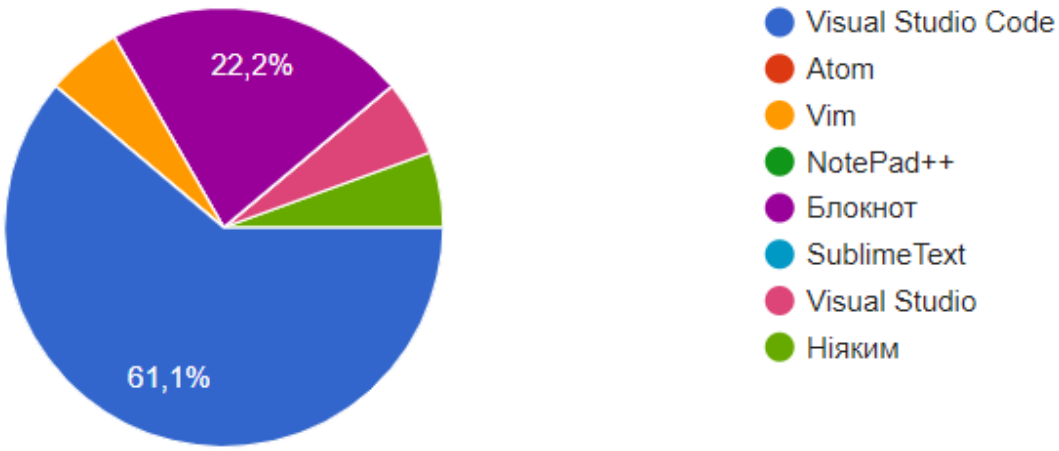


Рисунок 1.1 – Діаграма популярності інтелектуальних текстових редакторів

Також було відмічено, що найпопулярнішою мовою програмування серед опитаних фахівців стала мова JavaScript. На приведеному нижче рисунку 1.2 представлена статистика популярності мов програмування.

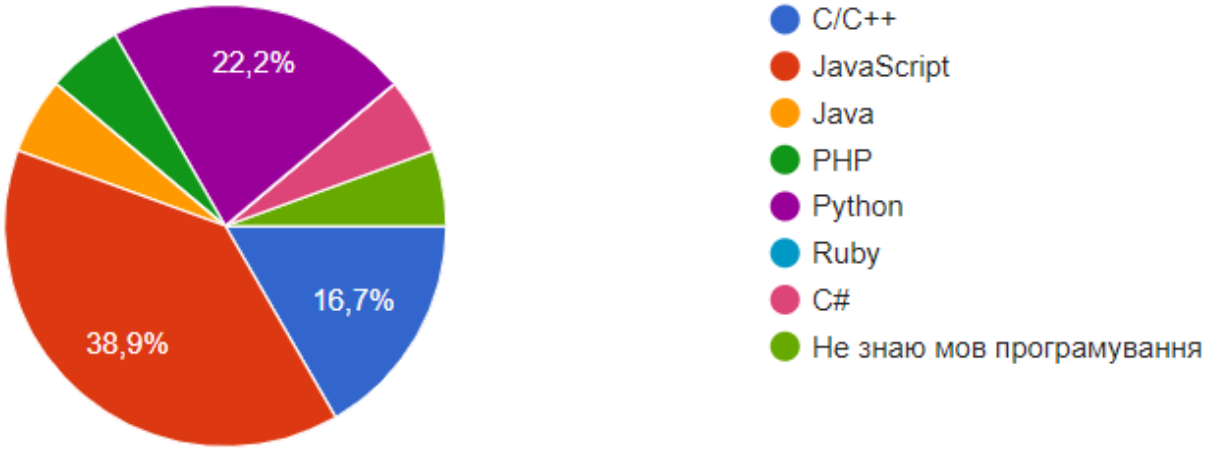


Рисунок 1.2 – Діаграма популярності мов програмування

Аналізуючи діаграму можна стверджувати, що найпопулярнішою з представлених мов стала мова JavaScript, на якій програмують майже 40% анкетованих

фахівців. Другою за популярністю стала мова Python, доля якої складає понад 20%. На третьому місці стоять мови програмування C/C++, на які приходиться більш ніж 16% відповідей.

Крім того, на рисунку 1.3 можемо побачити, що однією з найважливіших функцій редактора згідно з результатами анкетування стала функція можливості зіставлення двох документів для порівняння коду. Це можна побачити на графіку приведеному нижче, цю функцію відмітили важливою понад 72% фахівців, яким була представлена до заповнення анкета.

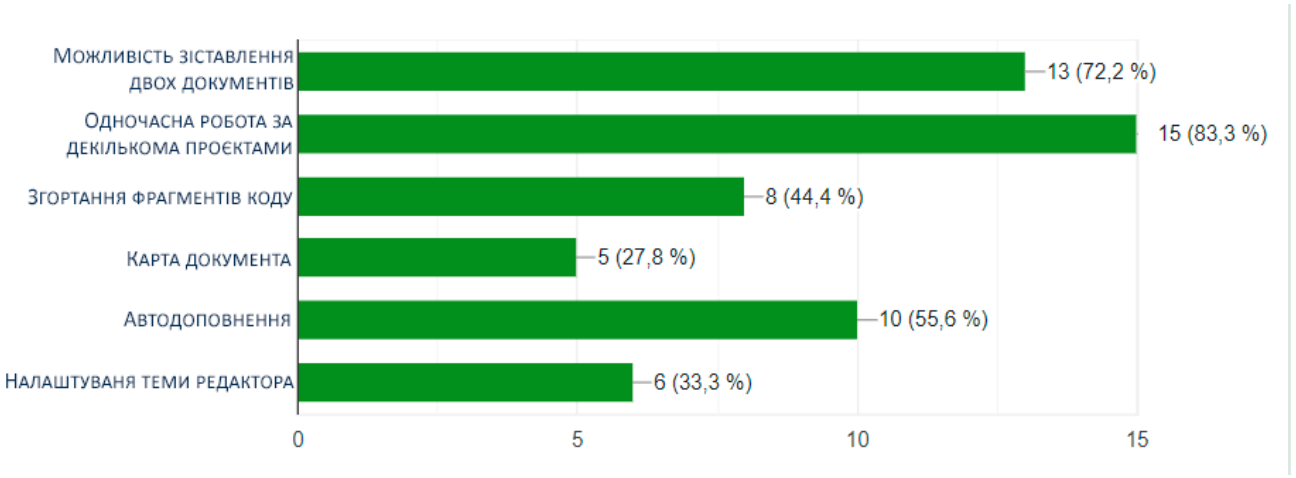


Рисунок 1.3 – Діаграма популярності функцій інтелектуального текстового редактора

Аналізуючи наступний графік на рисунку 1.4 можемо бачити залежність від рисунку 1.2. Маємо схожу статистику популярності вибраних мов програмування. Мова JavaScript займає перше місце і 40% від загальної кількості опитаних. Друге місце посідає мова програмування Python з долею більш ніж 25%. На третьому місці знаходяться мови програмування C/C++, за які віддали свої голоси більш ніж 10% опитаних фахівців.

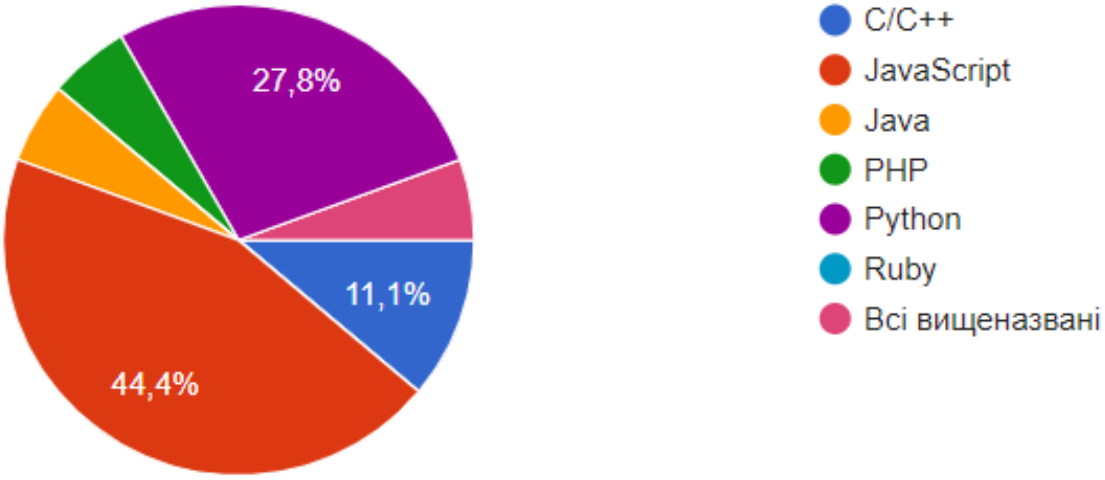


Рис. 1.4 – Діаграма популярності мов програмування, які буде підтримувати інтелектуальний текстовий редактор

1.3 Прототипування

Прототипування – це один із початкових етапів розробки, в ході якого створюється попередній дизайн програми або іншого проекту.

Під час прототипування створюється макет, що імітує взаємодію користувача з інтерфейсом проекту.

Прототип потрібен для презентації проекту замовнику та оцінки його юзабіліті. Тестування такого макету дозволяє заздалегідь виявити та усунути помилки, перш ніж вкладати гроші в розробку кінцевого дизайнерського рішення та коду.

Прототипування вирішує кілька важливих завдань.

Пошук найкращих ідей. Прототип робиться швидко, тому можна відразу підготувати кілька варіантів для тестування гіпотез, щоб потім вибрати найбільш вдалий. Це особливо актуально для стартапів.

Виявлення помилок. На етапі створення макету можна відстежити ключові недоліки майбутнього сайту або програми. На їх виправлення ви витратите менше часу, грошей та зусиль, ніж якби довелося вносити коригування в кінцевий продукт.

Оцінка юзабіліті. Розробка прототипів і тестування на них сценаріїв користувача – відмінна можливість якомога раніше перевірити, наскільки рішення зручне для користувачів.

Типи прототипів.

По глибині опрацювання прототипи бувають із високою та низькою деталізацією. Все залежить від кількості елементів у підсумковому варіанті.

По можливості взаємодії з макетом прототипи поділяються на статичні та інтерактивні. Статичні можна зобразити на папері схематично, а для інтерактивних варто використовувати графічні редактори.

Етапи прототипування.

Постановка цілі. Відбувається на зустрічі всіх учасників, серед яких клієнт, дизайнер, маркетолог, програміст, тобто всі, хто зацікавлений у успішності проекту. Чим чіткіше і точніше сформульовані цілі, тим легше висувати та перевіряти гіпотези для деталізації прототипу.

Проведення дослідження. Щоб створити якісний прототип, важливо вивчити бізнес клієнта, особливості його продукту та цільової аудиторії.

Формулювання гіпотез. Важливо зрозуміти, що ви хочете перевірити за допомогою прототипів. Не "подивитися, чи вдасться розмістити всі блоки на сайті", а "оцінити, наскільки користувачеві зручно буде вибрати товар і зробити замовлення в такий спосіб". Це допоможе зробити прототипування максимально ефективним.

Створення прототипу. З урахуванням результатів дослідження та сформульованих гіпотез створюється макет майбутнього сайту або програми.

На рисунку 1.5 представлено прототип інтерфейсу користувача.

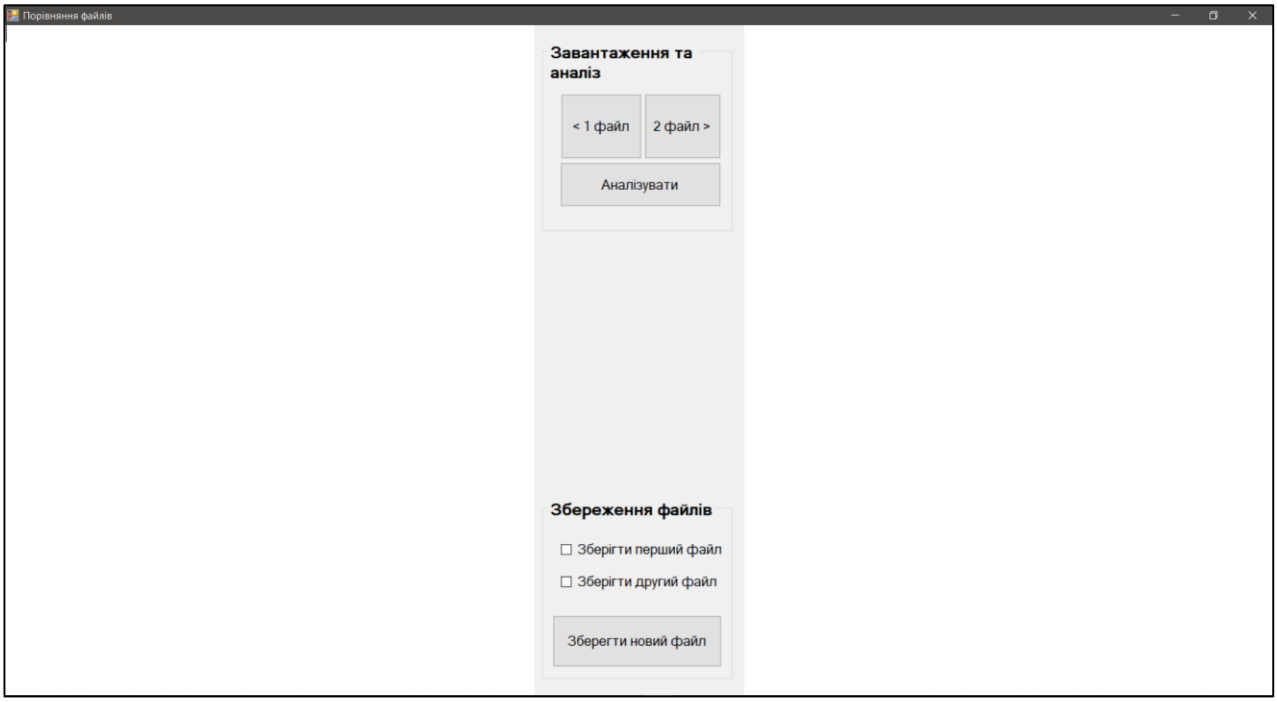


Рисунок 1.5 – Прототип інтерфейсу користувача

1.4 Огляд аналогів

За версією Stack Overflow [2] рейтинг найпопулярніших редакторів коду за 2021 рік представлений на рисунку 1.6

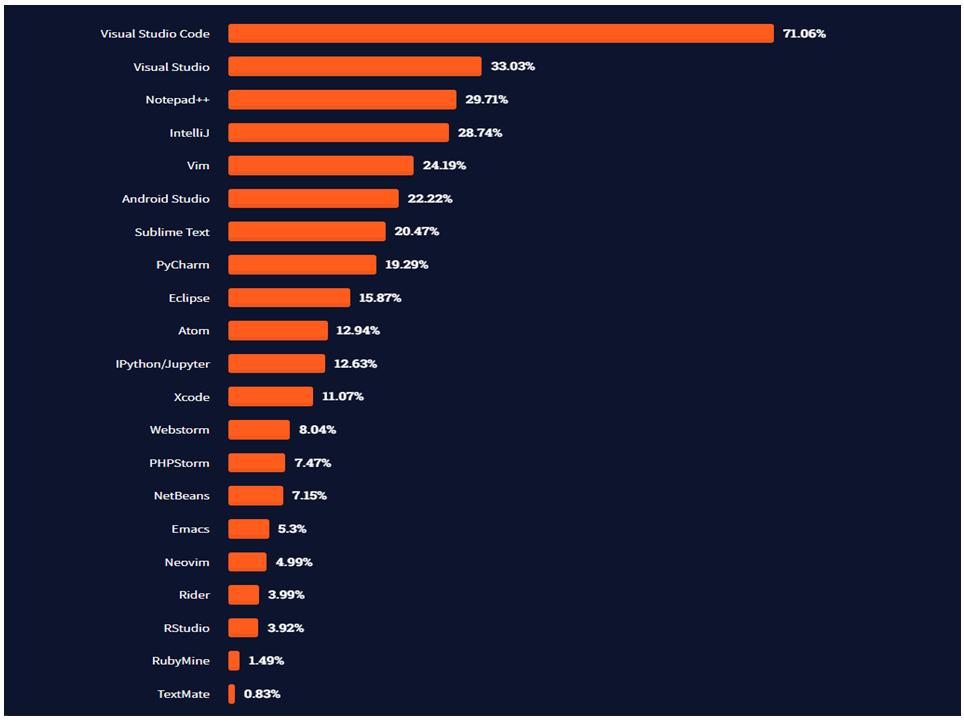


Рисунок 1.6 – Рейтинг найпопулярніших редакторів

Спираючись на представлений рейтинг було зроблено аналіз та описано 5 редакторів, які входять у десятку найпопулярніших, та підтримують принаймні декілька мов програмування.

Visual Studio Code – це легкий, але потужний редактор вихідного коду, доступний для Windows, macOS та Linux. Він поставляється з вбудованою підтримкою JavaScript, TypeScript і Node.js і має багату екосистему розширень для інших мов (наприклад, C++, C#, Java, Python, PHP, Go) і середовищ виконання (наприклад, .NET і Unity).

Функціонал:

– вбудовані інструменти інтеграції за Git, а також Visual Studio Team Services для швидкого тестування, складання, упаковки та розгортання різних типів додатків;

– підтримка майже всіх мов програмування;

– написання коду для конкретної задачі, з подальшою можливістю його інтеграції у проект;

– одночасна робота за декількома проектами;

– автодоповнення та автоматичне закриття скобок;

– функції відлагодження;

– можливість розділення інтерфейсу на дві панелі для порівняння коду;

– згортання фрагментів коду;

– карта документа.

Плюси:

– простота та гнучкість;

– багатофункціональність;

– розширювана бібліотека доповнень та готових рішень;

– безліч налаштувань як програми, так і інтерфейсу.

Мінус – інтерфейс та іконки підходять не для всіх.

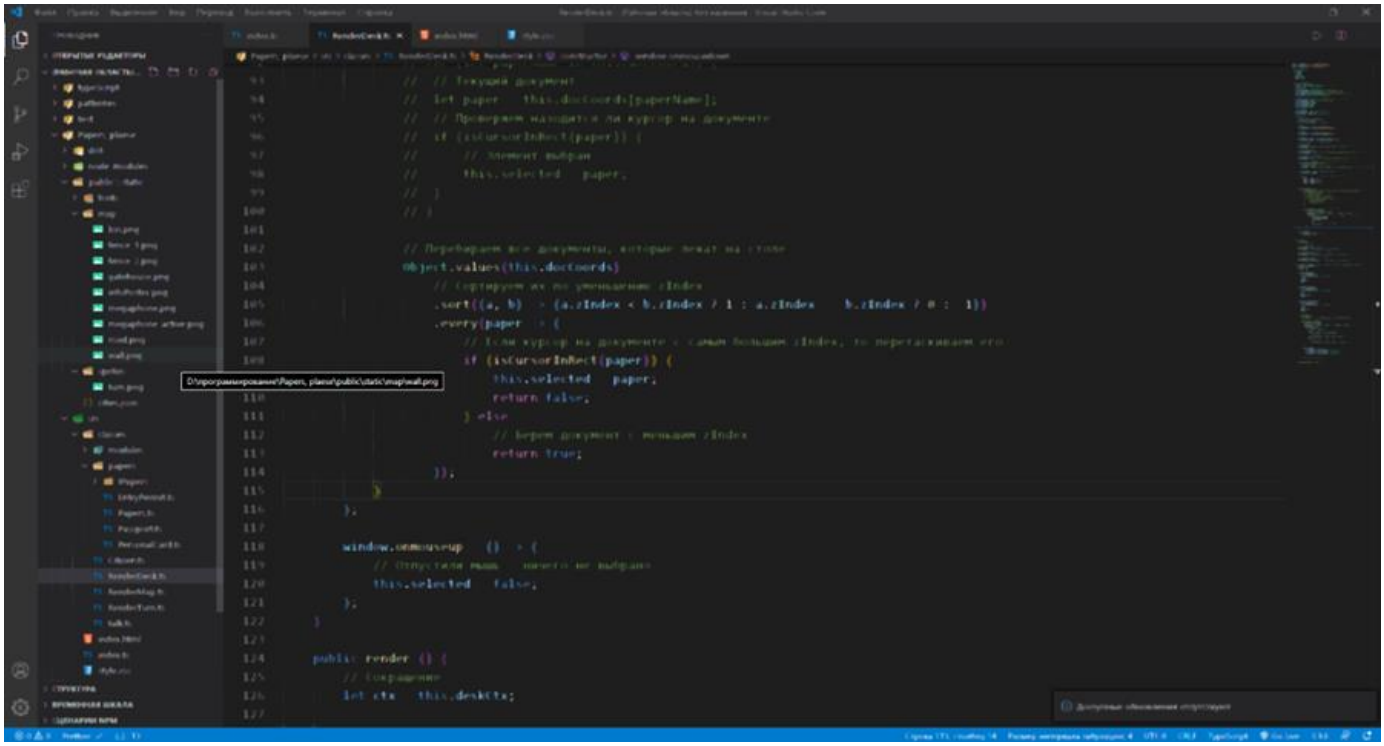


Рисунок 1.7 – Інтерфейс програми Visual Studio Code

На рисунку 1.7 представлений інтерфейс програми Visual Studio Code. Він складається з основної області для написання коду, зліва можемо побачити список файлів, які існують у проекті в зручній деревоподібній структурі. Зверху над основної області розташований список відкритих вкладок, тобто файлів проєкту, шо робить зручним та швидким перехід між ними ті їх редагування. З правого боку можна побачити невелику карту файлу, яка допомагає швидше перемішуватися по різних місцях у файлі.

Notepad++ – це безкоштовний редактор вихідного коду та заміна блокнота, який підтримує кілька мов.

Функціонал:

– підтримка та підсвічування синтаксису для поширених мов програмування;
– згортання фрагментів коду;
– карта документа;
– закладки;

– порівняння файлів;

– автодоповнення та автоматичне закриття скобок.

Плюси:

– підтримка вкладок;

– невеликий розмір та невелике споживання ресурсів;

– гнучкість налаштування;

– налаштування кольорових схем для кращого сприйняття коду.

Мінус – не дуже сучасний інтерфейс.

На рисунку 1.8 представлений інтерфейс програми Notepad++. Він складається з основної області для написання коду та вкладок, які відображують відкриті файли. На відміну від редактора Visual Studio Code, редактор Notepad++ не дозволяє відкрити весь проєкт, а дозволяє відкривати тільки окремі файли.

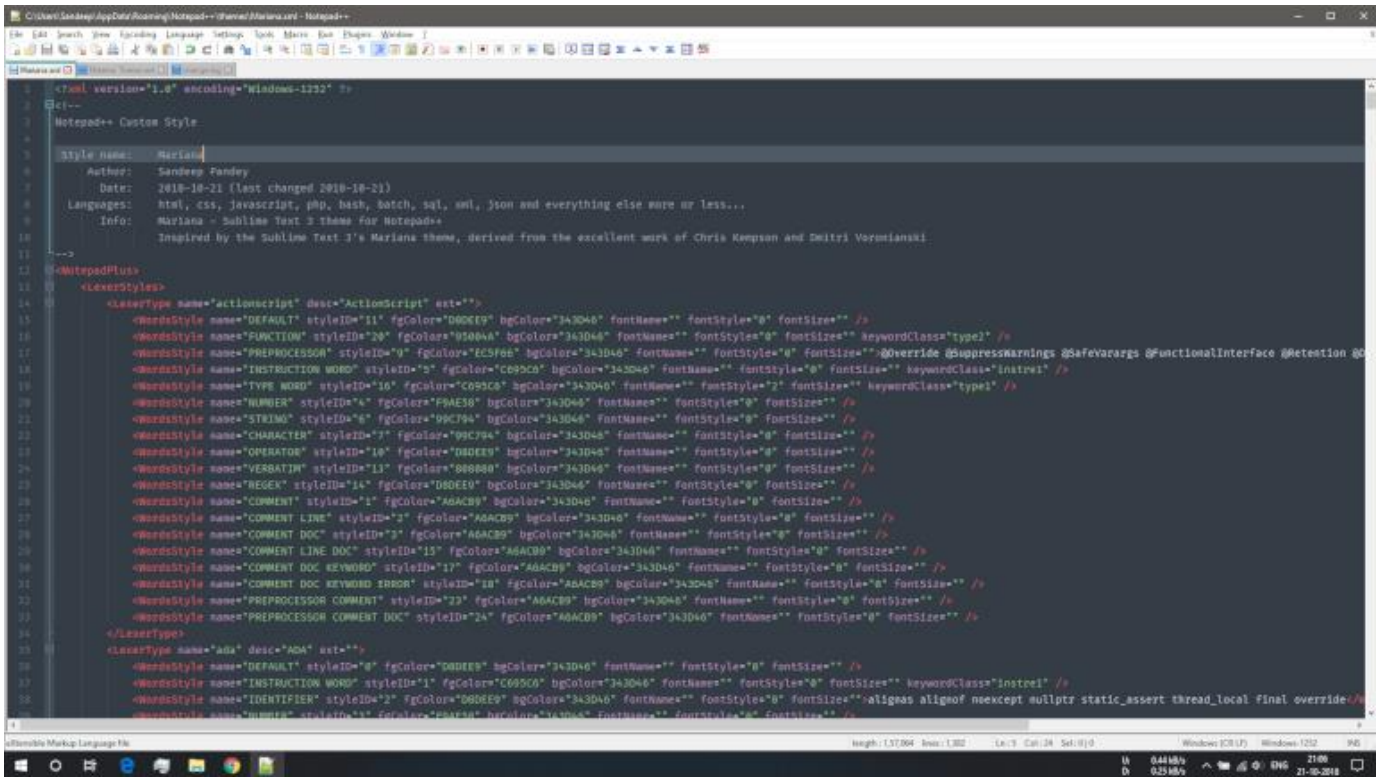


Рисунок 1.8 – Інтерфейс програми Notepad++

Vim – це текстовий редактор, який можна зручно налаштувати під себе завдяки розширенням та великому вибору інструментів для користувача. Він працює у терміналі, що дає можливість розробникам значно швидше виконувати завдання. Завдяки командним функціям Vim допомагає збільшити продуктивність роботи. Це вибір професіоналів, які використовують консолі для швидкого та зручного написання коду.

Функціонал:

– застосування двох основних режимів введення: командного (після запуску редактор знаходиться в ньому) і текстового(режим безпосереднього редагування тексту, аналогічний більшості «звичайних» редакторів);

– можливість автоматизації окремих операцій.

Плюси:

– велике число плагінів;

– робота з файлами великого розміру;

– не потрібна миша.

Мінуси:

– нестандартні hot key;

– застарілий;

– потребує попереднього навчання перед початком користування;

– потребує довгої конфігурації під користувача.

На рисунку 1.9 представлений інтерфейс програми VIM. Він складається з двох основних областей для редагування файлів, які розташовані поруч одна з одною та можуть бути збільшені або зменшені. Також зліва можна побачити меню відкритих файлів.

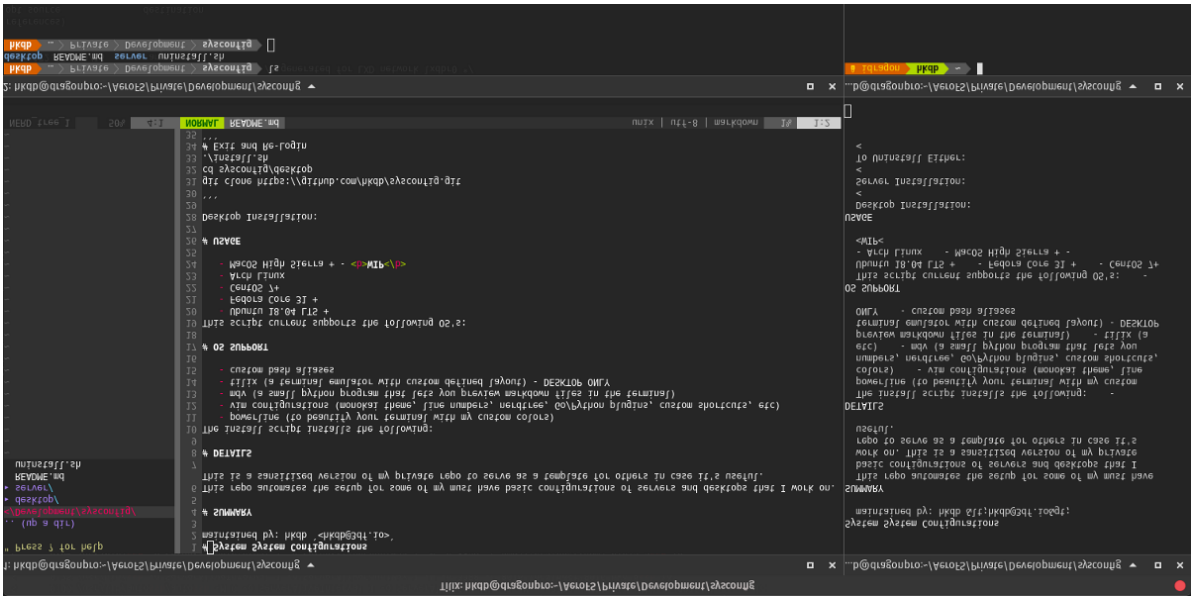


Рисунок 1.9 – Інтерфейс програми VIM

Sublime text – один з найпопулярніших текстових редакторів у світі. Він наповнений потужними функціями, такими як багаторядкове редагування, системи побудови для десятків мов програмування, пошук і заміна регулярних виразів тощо.

Функціонал:

– підтримка багатьох популярних мов програмування;

– зіставлення скобок допомагає визначити не правильне положення скобок та виділяє відповідні номера скобок;

– множинне виділення – дозволяє швидко заміняти строки коду.

Плюси:

– швидкість та чуйність користувацького інтерфейсу;

– кросплатформенність;

– зручне переключення між різними файлами;

– простота користування;

– різноманіття плагінів, які інтегруються в одному місці;

– повністю налаштовується користувачем у разі потреби.

Мінус – деякі плагіни в редакторі глючать.

На рисунку 1.10 представлений інтерфейс програми Sublime Text. Аналогічно описаним вище редакторам, має основну область для редагування коду у файлі, а також область зліва ідентичну області редактора Visual Studio Code, тобто відображає відкриті проєкти та їх вміст. Зверху також знаходиться невелика зручна панель, на якій відображено відкриті файли.

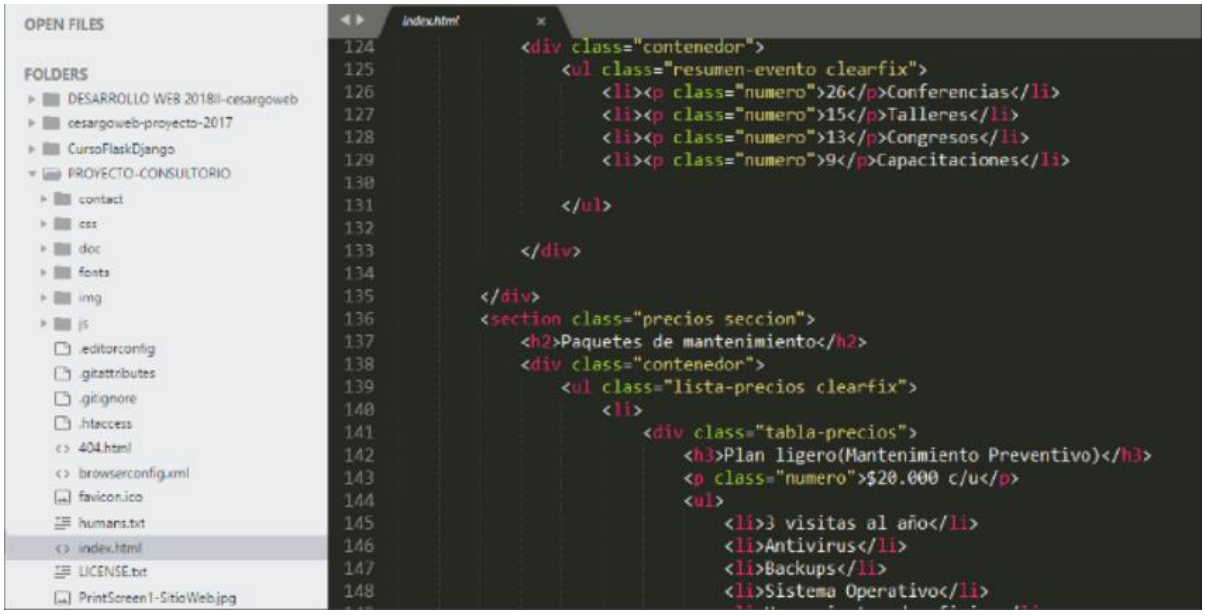


Рис. 1.10 – Інтерфейс програми Sublime Text

Atom – це функціональний текстовий редактор від розробників GitHub. Він підтримує безліч різноманітних розширень, завдяки яким його можна порівняти з реальним середовищем розробки. Інша особливість – платформа Electron, де тримається Atom. Вона включає Node.js і двигун від Chrome - такий інструментарій дозволяє розробляти програми для робочого столу на основі веб-технологій.

Функціонал:

– підсвічування синтаксису для всіх популярних мов: PHP, HTML, Json, SQL, XML, CSS, CoffeeScript, JavaScript, Java, C/C++, Go;

– зручна синхронізація Atom та Git;

– автоматичне форматування коду;

– згортання фрагментів коду;

– карта документа.

Плюси:

– кросплатформенність (працює на Mac, Linux, Windows);

– зручний пошук файлів у проекті (навіть за допомогою нечіткого пошуку;

– широкий вибір налаштувань теми редактора;

– вбудований менеджер пакетів ті їх різноманітність;

– дві командні строки – одна з яких запускає додаток, друга – диспетчер пакетів.

Мінуси:

– повільний запуск редактора;

– відбуваються збої на зависання файлів понад 10МБ.

На рисунку 1.11 представлено інтерфейс програми Atom. Як і представлені вище редактори, Atom має основну область для редагування коду, над якою розташовується панель, на якій показані відкриті файли. Зліва також присутня вкладка з мапою відкритого у редакторі проєкту. Справа можемо побачити панель, на якій можні вибрати пакети або плагіни у разі потреби.

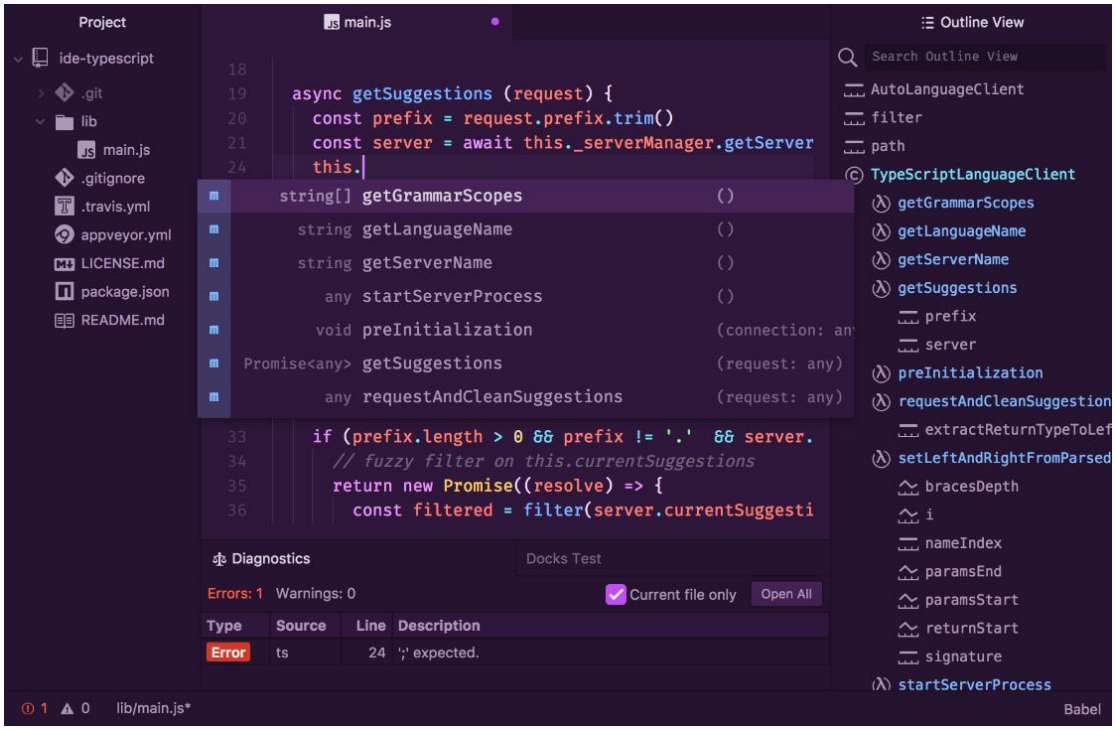


Рисунок 1.11 – Інтерфейс програми Atom

Висновки до розділу

У даному розділі дипломної роботи проведено опитування цільової аудиторії та виявлено найпопулярніші редактори коду, а також основні функції озвучених вище редакторів, на які звертають увагу користувачі під час вибору програми для подальшої роботи. Крім того проведено огляд найпопулярніших інтелектуальних редакторів коду, визначено їх сильні та слабкі сторони та наявність функціоналу у тій чи іншій програмі порівняно з вимогами користувачів. Спираючись на результати проведеного аналізу, можна зробити кілька висновків:

– представлені над ринком текстові редактори коду у своєму початковому варіанті немає достатнього набору функцій задоволення всіх запитів користувачів;

– більшість інтелектуальних текстових редакторів – це програми з відкритим кодом, що дозволяє розширювати функціонал за допомогою нових плагінів та їх підключення без впливу на продуктивність редактора під час його запуску та в процесі його роботи.

У процесі написання коду у користувача може з'явитися кілька версій одного і того ж документа і надалі встати питання - чим ці версії відрізняються. Оцінка різних версій документа, а також пошук та фіксування відмінностей у двох версіях – це один з найпопулярніших запитів користувачів, згідно з проведеним опитуванням. В основу даної роботи ліг процес розробки саме такого плагіна.

РОЗДІЛ 2. ПРОЄКТУВАННЯ

Проектування є основною та вирішальною фазою життєвого циклу програмного забезпечення, під час якого створюється та на 90 % набуває своєї остаточної форми програмного виробу. Ця фаза життя охоплює різні види діяльності проекту і може бути поділена на три основні етапи: конструювання, програмування та налагодження програмного виробу.

Конструювання програмного забезпечення зазвичай починається як тільки виявляються зафіксованими на папері деякі попередні цілі та вимоги до нього. На момент затвердження вимог робота у фазі конструювання буде у самому розпалі.

На цьому етапі здійснюють:

– функціональну декомпозицію вирішуваного завдання, на основі якого визначається архітектура функціональної системи;

– зовнішнє проектування ПЗ, що виражається у формі зовнішньої взаємодії його з користувачем;

– проектування бази даних, якщо це необхідно;

– проектування архітектури програмного забезпечення, тобто визначення об'єктів, модулів та їх сполучень.

Програмування починається вже у фазі конструювання, щойно стануть доступними основні специфікації на окремі компоненти програмного виробу, але не раніше затвердження угоди про вимоги.

На цьому етапі виконується робота, пов'язана зі збиранням програмного виробу. Вона полягає у докладному внутрішньому конструюванні програмного продукту, у створенні внутрішньої логіки кожного модуля системи, що потім виражається текстом конкретної програми.

Фаза програмування завершується, коли робітники закінчать документування, налагодження та компонування окремих частин програмного виробу.

Налагодження програмного забезпечення здійснюється після того, коли всі його компоненти будуть налагоджені окремо і зібрані в єдиний програмний продукт [3].

2.1 Зовнішнє проєктування

Зовнішнє проектування програмного забезпечення – це процес опису зовнішніх функцій проекту і очікуваної поведінки продукту, що розробляється з погляду зовнішнього по відношенню до нього спостерігача-користувача.

Мета цього процесу – конструювання зовнішніх взаємодій майбутнього програмного продукту із зовнішнім середовищем (зазвичай з користувачем) без конкретизації його внутрішнього пристрою.

Зовнішнє проектування програмного виробу виражається у формі зовнішніх специфікацій, призначених для широкої аудиторії, включаючи користувача (для перевірки та схвалення), авторів документації для користувача, всіх програмістів, що беруть участь у проекті, а також усіх тих, хто займатиметься тестуванням продукту.

Функціональне призначення – програмний продукт використовує різні версії документів користувача, або різних користувачів та полегшує формування нового документу на основі попередніх.

Експлуатаційне призначення – за допомогою розширення виконується зіставлення різних версій документів задля швидкого пошуку відмінностей та створення нового документу за бажанням користувача.

Функціональні вимоги

– перевірка не співпадаючих строк у документі;
– перевірка на наявність нових строк у документі.

Програма повинна забезпечити можливість керування документами, а саме:

– видалення непотрібних фрагментів коду з документу;
– додавання нових фрагментів коду до документу;
– збереження кінцевої версії документу.

В програмі необхідно реалізувати формування вихідного документу з урахуванням всіх змін та редагувань вхідних документів користувачем.

Вхідні данні – два текстових файлу, які містять код, або фрагмент коду.

Розширення вхідних документів повинні бути однаковими.

Схема роботи програми:

1. Прочитати два вхідних файли, вибраних користувачем.

2. Перевірити файли на збіг строк, строки які не збігаються виділити кольором.

3. Перевірити документ на додані фрагменти коду, виділити їх червоним у другому документі.

4. Зберегти відредагований користувачем файл.

Вихідні дані – кінцевий відредагований користувачем файл.

В ході роботи програми створюється файл, з таким же розширенням, як і два попередні.

Опис зовнішнього інформаційного середовища. Вхідні файли вибираються через вікно вибору файлу за допомогою кнопки “Завантажити файли”. Результати обробки документів представляються користувачу у вигляді двох областей з текстами двох завантажених файлів. Відредагований файл можна зберегти натиснувши на кнопку “Зберегти файл” через вікно збереження файлу.

2.2 Внутрішнє проєктування

2.2.1 Проєктування архітектури системи

Ідентифіковані сутності: аналіз файлів, алгоритм пошуку змін в строках файлів, алгоритм пошуку видалених фрагментів коду, алгоритм пошуку доданих фрагментів коду, кінцевий документ для збереження. Додатково: інтерфейс користувача у вигляді екранної форми.

Ідентифіковані обов’язки:

– аналіз файлів – виконання порівняння двох файлів за вказаними зовнішніми специфікаціями;

– алгоритм пошуку змін в строках документу – реалізація алгоритму пошуку

неспівпадаючих рядків у вхідних файлах;

– алгоритм пошуку доданих фрагментів коду – реалізація алгоритму визначення фрагментів коду із другого файлу, які не існують у першому файлі;

– кінцевий документ для збереження – збереження документа, відредагованого користувачем;

– форма – завантаження вхідних файлів, перегляд результатів порівняння файлів, виконання команди збереження нового файлу.

Таблиця 2.1 – Сутності, атрибути та методи

Сутність	Атрибути	Методи
Аналіз файлів	Два файли з однаковим розширенням Параметри аналізу файлів	Запустити
Алгоритм пошуку змін в строках документу	Масив строк першого файлу Масив строк другого файлу	Метод порівняння строк
Алгоритм пошуку доданих фрагментів коду	Масив строк першого файлу Масив строк другого файлу	Метод пошуку нових строк
Кінцевий документ для збереження	Масив строк відредагований користувачем для запису у файл	Записати дані в файл
Форма	Поле для виведення тексту першого файлу Поле для виведення тексту другого файлу Кнопка завантаження файлів Кнопка збереження кінцевого файла	Завантажити файли Зберегти в файл

Моделювання розподілу обов’язків у системі

Множини класів, які працюють спільно для досягнення деякої поведінки:

– форма, аналіз файлів – завантаження файлів для аналізу, та його запуск, перегляд результатів;

– аналіз файлів, алгоритм пошуку змін в строках документу – знаходження відмінностей у документі;

– аналіз файлів, алгоритм пошуку видалених фрагментів коду – знаходження відмінностей у документі.

2.2.2 Проєктування інтерфейсу користувача

Інтерфейс користувача (UI – англ. user interface) являє собою сукупність засобів і методів, за допомогою яких користувач взаємодіє з різними пристроями та апаратурою. Іншими словами, це той набір кнопок, посилань, форм, діалогових вікон, іконок, піктограм, банерів, повзунків та стрічок прокручування, за допомогою якого користувач керує продуктом

Проектування інтерфейсу користувача – це створення тестової версії програми. Це початковий етап розробки інтерфейсу користувача, коли розподіляються функції програми по екранах, визначаються макети екранів, вміст, елементи управління та їх поведінка [4].

Користувач при поводженні з інтерфейсом повинен уявити собі, яка інформація про завдання, що виконується, у нього існує, і в якому стані знаходяться засоби, за допомогою яких він вирішуватиме дане завдання. Ефективність роботи користувача та його інтерес забезпечує правильно сформульована методика розробки та проектування інтерфейсу користувача.

Саме тому необхідно велику увагу приділяти процесу побудови інтерфейсів (UI) і вибудовуванню досвіду в цілому (UX). Проектування UI – це не разова фаза проекту, це безперервний ітераційний процес, до якого залучені бізнес-користувачі, UX-інженери, дизайнери та програмісти.

Інтерфейс – лише половина у взаємодії із системою, інша половина – людина, користувач. Для хорошої роботи інтерфейсу потрібно точно знати, що саме у будь-який конкретний момент користувач сприймає в інтерфейсі, про що думає, чого хоче досягти.

Інтерфейси є основою взаємодії сучасних інформаційних систем. Якщо інтерфейс будь-якого об'єкта (персонального комп'ютера, програми, функції) не змінюється (стабільний, стандартизований), це дозволяє модифікувати сам об'єкт, не перебудовуючи принципи його взаємодії з іншими об'єктами [5].

Переваги гарного ПІ:

– підвищення конкурентоспроможності;

– зниження вартості розробки;

– збільшення аудиторії товару;

– зменшення витрат на навчання та підтримку користувачів;

– зменшення втрат продуктивності працівників при впровадженні системи та швидше відновлення втраченої продуктивності;

– доступність функціональності системи для максимальної кількості користувачів;

– зниження ризику помилок.

Відмінні риси якісного інтерфейсу:

– стильова гнучкість – можливість використовувати різні інтерфейси з одним і тим самим додатком, на практиці реалізується у вигляді набору «skins», для web-інтерфейсів – за допомогою таблиці стилів, у тому числі можливість у виборі користувачем власних установок ПІ (колір, ікони) , підказки та ін.);

– спільне нарощування функціональності – можливість розвивати програму без руйнування (тобто залишаючись у межах) існуючого інтерфейсу;

– масштабованість – можливість легко налаштовувати та розширювати як інтерфейс, так і сам додаток зі збільшенням кількості користувачів, робочих місць, обсягу та характеристик даних;

– адаптивність до дій користувача – програма повинна допускати можливість введення даних та команд безліччю різних способів (клавіатура, миша, інші пристрої) та багатоваріативність доступу до прикладних функцій (ікони, гарячі клавіші, меню тощо).

Крім цього програма повинна враховувати можливість переходу та повернення від вікна до вікна, від режиму до режиму та правильно обробляти такі ситуації;

– незалежність в ресурсах – для створення інтерфейсу користувача повинні надаватися окремі ресурси, спрямовані на зберігання та обробку даних, необхідних для підтримки користувача (користувацькі словники, контекстнозалежні списки, набори даних за замовчуванням або за останнім запитом, історії запитів тощо);

– кросплатформенність - при переході на іншу апаратну (програмну) платформу повинен здійснюватися автоматичне перенесення і інтерфейсу користувача, і кінцевого додатка;

– мультимедійність - сукупність всіх видів інформації (графічної, звукової, відео).

В якості засобу формалізації використано діаграму станів, яка представлена на рисунку 2.1, для більш зручного розуміння сценарію діалогу користувача з системою. В якості засобу формалізації використаємо діаграму станів. На діаграмі стани, позначені кругом та кругом в колі, є початковим та заключним станами відповідно. Вони позначають початок та завершення роботи з програмою. Вихід з програми можливий з будь-якого стану, проте, зважаючи на функціональне призначення програми, перехід у заключний стан на діаграмі представлено в одному екземплярі. Прямокутниками з заокругленими кутами позначені стани користувача.

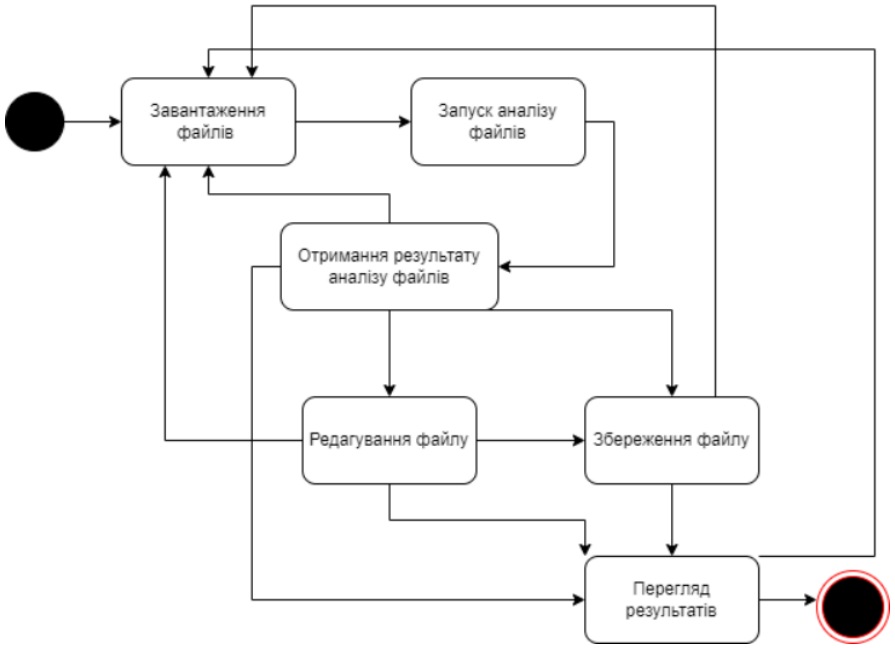


Рисунок 2.1 – Діаграма станів

2.2.3 Проєктування динаміки системи

Динаміка системи об‘єктно-орієнтованих систем проєктується на основі аналізу застосування системи (Use Case Diagram) та первинної класифікації. Аналізуючи кожен варіант застосування, треба визначити, об'єкти яких класів повинні створюватися, бути активними та взаємодіяти між собою. При цьому визначаються конкретні методи класів, які необхідні для реалізації того чи іншого варіанту використання програми.

Результати проєктування динаміки системи можуть бути представлені за допомогою діаграми діяльності (Activity Diagram).

Діаграми діяльності можна використовувати на всіх етапах розробки програмного забезпечення та для різних цілей. І оскільки вони дуже схожі на блок-схеми, вони зазвичай популярніші, ніж інші типи діаграм UML. Основною відмінністю діаграм діяльності від блок-схем є активна підтримка паралельних процесів, що пояснює застосування діаграми діяльності для моделювання потоків робіт

Діаграма активності UML дозволяє детальніше візуалізувати конкретний випадок використання. Це поведінкова діаграма, що ілюструє потік діяльності через систему.

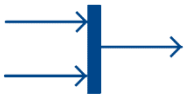
Діаграми активності UML можуть бути використані для відображення потоку подій у бізнес-процесі. Вони можуть бути використані для вивчення бізнес-процесів з метою визначення їх потоку та вимог.

У UML вказано набір символів та правил для побудови діаграм активності. Нижче наведено символи діаграми, що часто використовуються, з поясненнями.

Таблиця 2.2 – Символи діаграми діяльності

Символ	Ім'я	Використати
1	2	3
	Пуск/ початковий вузол	Використовується для представлення відправної точки або початкового стану діяльності
	Дія / Стан дії	Використовується для представлення діяльності процесу
	Дія	Використовується для представлення виконуваних підрайонів діяльності
	Потік управління / Край	Використовується для представлення потоку управління від однієї дії до іншої
	Потік об'єкта/краї управління	Використовується для відображення шляху руху об'єктів за активністю
	Кінцевий вузол активності	Використовується для позначення кінця всіх контрольних потоків у рамках діяльності

Закінчення таблиці 2.2

1	2	3
	Потік кінцевий вузол	Використовується для позначення кінця одного потоку керування
	Вузол прийняття рішень	Використовується для представлення умовної точки відгалуження з одним входом та кількома виходами
	Вузол злиття	Використовується для представлення злиття потоків. Він має кілька входів, але один вихід
	Виделка	Використовується для представлення потоку, який може розгалужуватися на два і більше паралельних потоку
	Злиття	Використовується для представлення двох входів, які поєднуються в один вихід
	Надсилання сигналу	Використовується для подання дії щодо надсилання сигналу на приймальну діяльність
	Отримання сигналу	Використовується для позначення того, що отримано сигнал
	Примітка/ коментар	Використовується для додавання відповідних коментарів до елементів

На рисунку 2.2 представлена діаграма діяльності, яка представляє дії для реалізації прецеденту отримання проаналізованих файлів. Ініціатором прецеденту-сценарію дій – є користувач як зовнішній об’єкт системи.

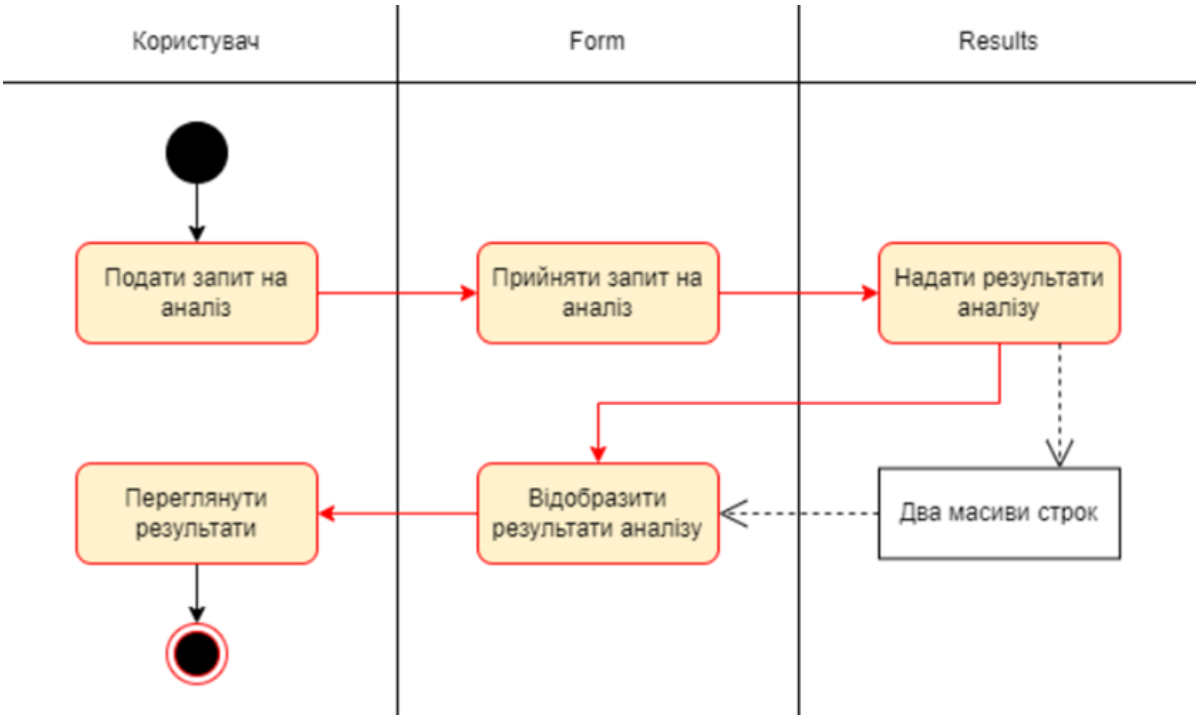


Рисунок 2.2 – Діаграма діяльності

Висновки до розділу

У цьому розділі була проведена робота із зовнішнього та внутрішнього проектування програмного засобу.

У ході роботи були описані зовнішні функції проекту та очікування користувачів у результаті реалізації цього проекту. Також описано загальну схему роботи програми. Внаслідок внутрішнього проектування були визначенні складові системи (програми), яка розробляється, їх структури та поведінки. Крім того, було виявлено особливості інтерфейсу користувача, визначено відмінні риси якісного інтерфейсу. Процес взаємодії користувача із системою відображено на діаграмі.

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМИ

3.1 Засоби опису алгоритмічних структур

Алгоритм – це послідовність команд, призначена виконавцю, в результаті виконання якої він має вирішити поставлене завдання. Алгоритм повинен описуватися формальною мовою, що виключає неоднозначність тлумачення.

Алгоритм повинен мати певні властивості. Найважливіші властивості алгоритмів:

– дискретність. Процес розв'язання завдання має бути розбитий на послідовність окремих кроків — простих дій, які виконуються одна одною у порядку. Кожен крок називається командою (інструкцією). Тільки після завершення однієї команди можна перейти до наступної;

– кінцівка. Виконання алгоритму має завершитись за кінцеве число кроків; при цьому має бути отриманий результат;

– зрозумілість. Кожна команда алгоритму має бути зрозумілою виконавцю. Алгоритм повинен містити ті команди, які входять до системи команд його виконавця;

– визначеність (детермінованість). Кожна команда алгоритму має бути точно та однозначно визначена. Також однозначно має бути визначено яка команда виконуватиметься на наступному кроці. Результат виконання команди не повинен залежати від жодної додаткової інформації. У виконавця має бути можливості прийняти самостійне рішення (тобто. він виконує алгоритм формально, не вникаючи у сенс). Завдяки цьому будь-який виконавець, що має необхідну систему команд, отримає той самий результат на підставі одних і тих же вихідних даних, виконуючи один і той же ланцюжок команд;

– масовість. Алгоритм призначений для вирішення не однієї конкретної задачі, а цілого класу задач, що визначається діапазоном можливих вхідних даних [8].

Способи представлення алгоритмів:

– словесна запис (природною мовою). Алгоритм записується у вигляді послідовності пронумерованих команд, кожна з яких є довільним викладом дії;

– блок-схема (графічне зображення). Алгоритм представляється за допомогою спеціальних значків (геометричних фігур) блоків;

– формальні алгоритмічні мови. Для запису алгоритму використовується спеціальна система позначень (штучна мова, яка називається алгоритмічною);

– псевдокод. Запис алгоритму на основі синтезу алгоритмічного та звичайного мов. Базові структури алгоритму записуються суворо за допомогою елементів деякої базової алгоритмічної мови.

Існують різні графічні засоби описування алгоритмічних структур.

Блок-схема – наочний спосіб представлення алгоритму. Блок-схема відображається у вигляді послідовності пов'язаних між собою функціональних блоків, кожен з яких відповідає виконанню однієї або кількох дій. Певному типу дії відповідає певна геометрична фігура блоку. Лінії, що з'єднують блоки, визначають черговість виконання дій. За замовчуванням блоки з'єднуються зверху донизу та зліва направо. Якщо послідовність виконання блоків має бути іншою, використовуються спрямовані лінії (стрілки).

Для представлення алгоритмів у даній роботі було вибрано блок схеми.

На рисунку 3.1 представлено блок-схему алгоритму фарбування синтаксису для більш комфортного сприйняття завантажених файлів.

На рисунках 3.2 та 3.3 представлено блок-схему алгоритму для розфарбування змінених та доданих фрагментів коду.

Спочатку створюються масиви строк двох завантажених файлів.

Далі аналізуються строки файлів та у відповідні кольори фарбуються змінені та додані строки.

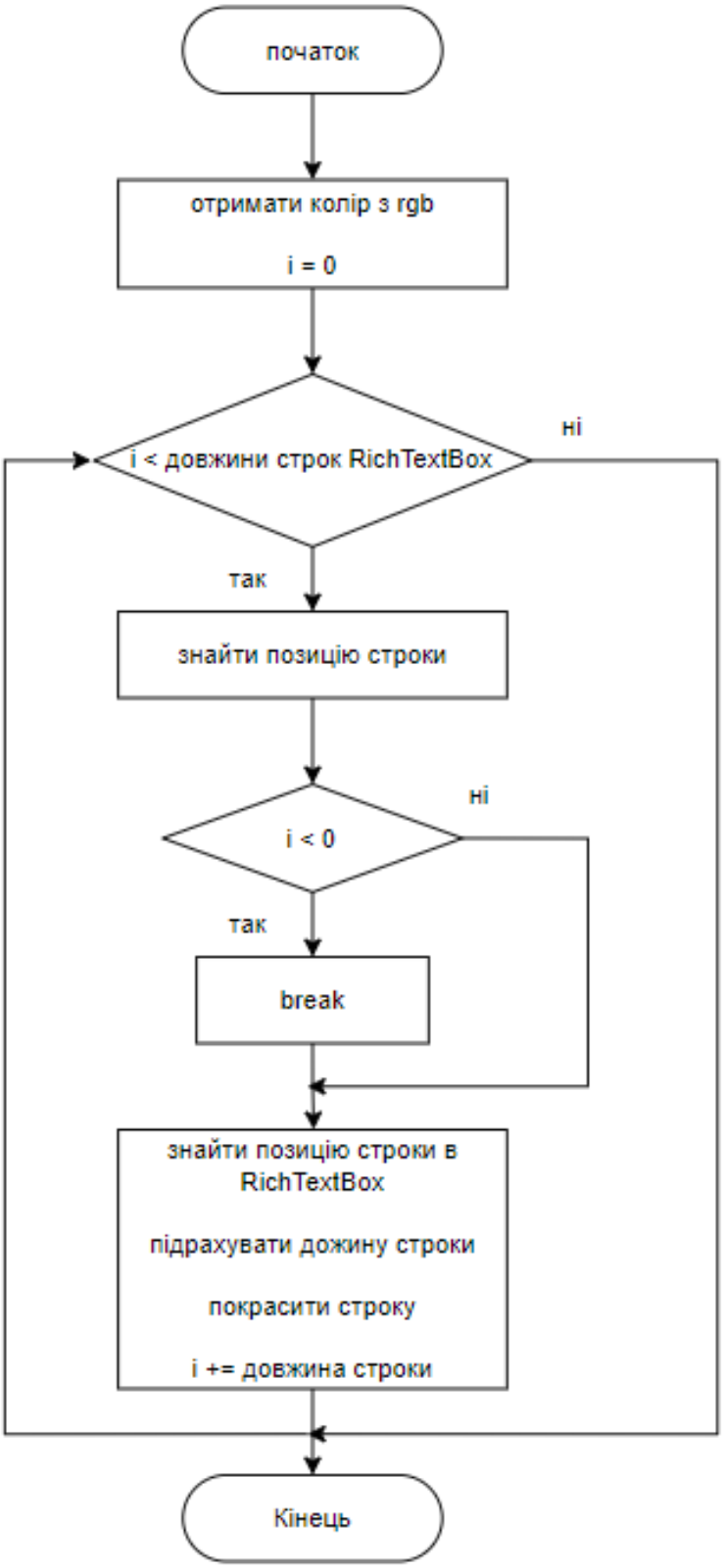


Рисунок 3.1 – Блок-схема алгоритму фарбування синтаксису

Рисунок 3.2 – Блок-схема початок алгоритму фарбування доданих та змінених частин коду

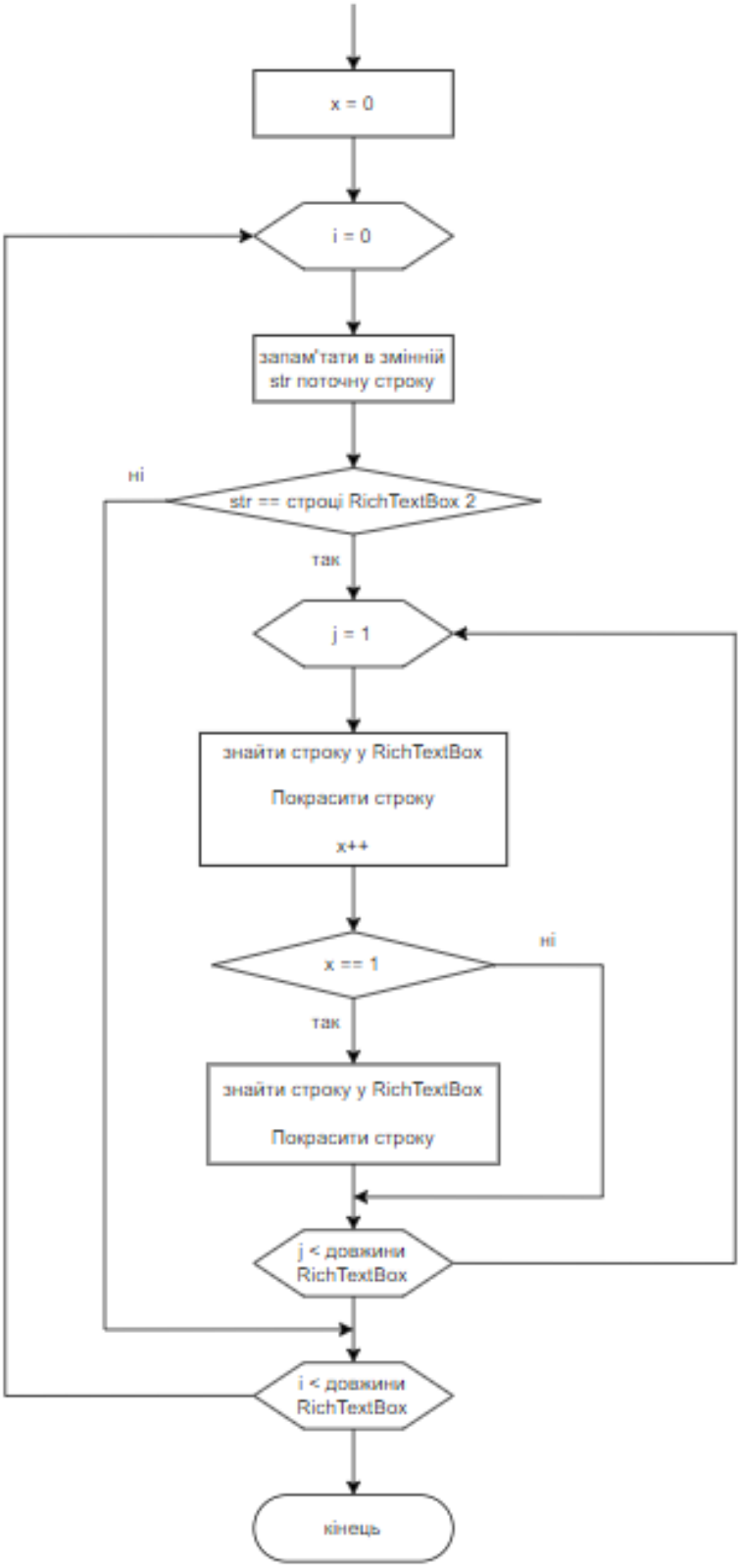


Рисунок 3.3 – Блок-схема закінчення алгоритму фарбування доданих та змінених частин коду

3.1 Написання коду

Для завантаження документів до програми використано клас OpenFileDialog.

Цей клас дозволяє перевірити, чи існує файл та відкрити його. Властивість ShowReadOnly визначає, чи відображається прапорець у діалоговому вікні тільки для читання. Властивість ReadOnlyChecked вказує, чи встановлено прапорець лише для читання.

Більшість основних функціональних можливостей цього класу знаходиться в FileDialog класі [11].

Таблиця 3.1 – Властивості класу OpenFileDialog

Властивості	Опис
1	2
AddExtension	Повертає або задає значення, що визначає, чи автоматично додає діалогове вікно розширення до імені файлу, якщо користувач опускає дане розширення.
AutoUpgradeEnabled	Повертає або задає значення, яке вказує на те, чи повинен цей FileDialog екземпляр автоматично оновлювати зовнішній вигляд і поведінку під час запуску у Windows Vista.
CanRaiseEvents	Повертає значення, що показує, чи компонент може викликати подію.
CheckFileExists	Повертає або задає значення, яке вказує, чи відображається попередження у діалоговому вікні, якщо користувач вказує неіснуюче ім'я файлу.
CheckPathExists	Повертає або задає значення, яке вказує на те, чи відображає діалогове вікно попередження, якщо користувач вказує неіснуючий шлях.

Продовження таблиці 3.1

1	2
ClientGuid	Отримує або задає GUID, що пов'язує з цим станом діалогу. Як правило, стан, такий як остання відвідана папка, а також розташування та розмір діалогового вікна, зберігається з урахуванням імені файлу, що виконується. При вказівці GUID програма може мати різні стани для різних версій діалогового вікна в тому ж додатку (наприклад, діалогове вікно імпорту і діалогове вікно відкриття). Ця функція недоступна, якщо програма не використовує стилі оформлення або якщо для AutoUpgradeEnabled встановлено значення false.
Container	Повертає об'єкт IContainer, який містить колекцію Component.
CustomPlaces	Отримує колекцію розміщень користувача для цього екземпляра FileDialog.
DefaultExt	Повертає або визначає розширення імені файлу за промовчанням.
DereferenceLinks	Повертає або задає значення, що вказує, чи діалогове вікно повертає розташування файлу, представленого ярликом, або повертає розташування самого ярлика (.lnk).
DesignMode	Повертає значення, що вказує, чи даний компонент Component в режимі конструктора в даний час.
Events	Повертає список обробників подій, які прикріплені до об'єкта Component.
FileName	Повертає або задає рядок, який містить ім'я файлу, вибраного в діалоговому вікні.
FileNames	Повертає імена всіх вибраних файлів у діалоговому вікні.

Продовження таблиці 3.1

1	2
Filter	Повертає або задає поточний рядок фільтра імен файлів, який визначає варіанти, доступні в полі діалогового вікна "Зберегти як тип файлу" або "Файли типу".
FilterIndex	Повертає або задає індекс фільтра, вибраного зараз у діалоговому вікні файлу.
InitialDirectory	Повертає або задає початкову папку, що відображається діалоговим вікном файлу.
Instance	Повертає дескриптор екземпляра обробника Win32 для програми.
Multiselect	Отримує або задає значення, яке вказує на те, чи можна в діалоговому вікні вибирати кілька файлів.
Options	Отримує значення ініціалізації класу FileDialog.
ReadOnlyChecked	Отримує або вказує, чи встановлено прапорець доступності лише для читання.
RestoreDirectory	Отримує або задає значення, яке вказує, чи відновлює діалогове вікно раніше вибраний каталог як поточний каталог перед закриттям.
SafeFileName	Повертає ім'я та розширення файлу, вибраного у діалоговому вікні. Ім'я файлу не містить відомостей про шлях.
SafeFileNames	Повертає масив імен та розширень файлів для всіх вибраних у діалоговому вікні файлів. Імена файлів не містять відомостей про шлях.
ShowHelp	Повертає або задає значення, яке визначає, чи відображається кнопка Довідка в діалоговому вікні роботи з файлами.

Закінчення таблиці 3.1

1	2
ShowReadOnly	Отримує або задає значення, яке вказує, чи є у діалоговому вікні прапорець "доступно лише для читання".
Site	Отримує або визначає ISite об'єкта Component.
SupportMultiDottedExtensions	Повертає або задає значення, що визначає, чи діалогове вікно підтримує відображення та збереження файлів, які містять кілька розширень імені.
Tag	Отримує або задає об'єкт, який містить дані елемента керування.
Title	Повертає або вказує заголовок діалогового вікна файлу.
ValidateNames	Повертає або задає значення, яке вказує на те, чи приймає діалогове вікно тільки допустимі імена файлів Win32.

Приклади використання функціональних можливостей у програмі наведено нижче.

```
openFileDialog.InitialDirectory = "c:\\Desktop";  // Повертає або задає початкову папку, що відображається діалоговим вікном файлу.
openFileDialog.Filter = "txt files (*.txt)|*.txt|All files (*.*)|*.*"; // Повертає або задає поточний рядок фільтра імен файлів, який визначає варіанти, доступні в полі діалогового вікна "Зберегти як тип файлу" або "Файли типу".
openFileDialog.FilterIndex = 2; // Повертає або задає індекс фільтра, вибраного зараз у діалоговому вікні файлу.
openFileDialog.RestoreDirectory = true; // Отримує або задає значення, яке вказує, чи відновлює діалогове вікно раніше вибраний каталог як поточний каталог перед закриттям.
```

Висновки до розділу

В даному розділі дипломної роботи було розроблено основну логіку роботи програми, а також створено та представлено алгоритми, необхідні для роботи програми та окремих її складових.

Було знайдено найкращі алгоритмічні рішення для цілей та задач проєкту, а також перетворено алгоритми у форму зручну для сприйняття.

Алгоритми були представлені в зручній формі у вигляді блок схем, що в подальшому полегшує їх використання при написанні коду програми.

РОЗДІЛ 4. ТЕСТУВАННЯ ТА НАЛАГОДЖЕННЯ

Тестування програмного забезпечення – це метод перевірки відповідності фактичного програмного продукту очікуваним вимогам, також необхідний, щоб переконатися, що продукт не містить дефектів. Має на увазі виконання попередньо визначених алгоритмів з використанням ручних або автоматизованих інструментів для оцінки одного або декількох властивостей, що цікавлять. Метою тестування є виявлення помилок, прогалин або відсутніх вимог, заданих на етапі проектування продукту.

Методи тестування програмного забезпечення:

«Біла скринька» – коли маємо доступ до коду, і ми його тестуємо, читаємо сам код (статичне тестування);

«Чорна скринька» – коли не знаємо, як система влаштована всередині, немає доступу до коду або не вміємо його читати, і тому орієнтуємось лише на зовнішню поведінку чи ТЗ.

4.1 Методи тестування програмного забезпечення

За знанням системи виділяють такі види тестування програмного забезпечення:

– біла скринька;

– чорна скринька.

Тестування білої скриньки – це метод тестування, який перевіряє внутрішнє функціонування системи. У цьому методі тестування ґрунтується на охопленні операторів коду, гілок, шляхів або умов. Тестування білої скриньки вважається тестуванням низького рівня. Це також називається скляним, прозорим, прозорим чи кодовим тестуванням. Метод тестування білої скриньки передбачає, що шлях логіки в модулі чи програмі відомий.

При тестуванні в «чорній скриньці» тестер не має інформації про внутрішню роботу програмної системи. Тестування чорної скриньки – це високий рівень тестування, який фокусується на поведінці програмного забезпечення. Це включає тестування з погляду зовнішнього або кінцевого користувача. Тестування чорної

скриньки може застосовуватися практично для будь-якого рівня тестування програмного забезпечення: юніт, інтеграція, система та приймання.

Порівняння методів тестування білої та чорної скриньки наведено у таблиці 4.1.

Таблиця 4.1 – Порівняння методів тестування

Параметр	Тестування чорної скриньки	Тестування білої скриньки
1	2	3
Визначення	Це метод тестування, який використовується для тестування програмного забезпечення без знання внутрішньої структури програми або програми	Це підхід до тестування, у якому внутрішня структура відома тестеру
Кличка	Він також відомий як кероване даними, блочне тестування, тестування даних та функціональне тестування	Це також називається структурне тестування, тестування прозорого боксу, тестування на основі коду або тестування прозорого боксу
База тестування	Тестування ґрунтується на зовнішніх очікуваннях; внутрішня поведінка програми невідома	Внутрішня робота відома і тестер може тестувати відповідно
Назва	Цей тип тестування ідеально підходить для більш високих рівнів тестування, таких як тестування системи, приймальне тестування	Тестування найкраще підходить для нижчого рівня тестування, таких як модульне тестування, інтеграційне тестування

Продовження таблиці 4.1

1	2	3
Знання програмування	Знання програмування не потрібні для тестування методом чорної скриньки	Знання з програмування необхідні для проведення тестування меттодом білої скриньки
Використання знань	Знання про використання не вимагають тестування методом чорної скриньки.	Повне розуміння потребує реалізації тестування методом білого ящика.
Автоматизація	Тест та програміст залежать один від одного, тому складно автоматизувати	Тестування білої скриньки легко автоматизувати
Завдання	Основна мета цього тестування – перевірити,яка функціональність системи, що тестується	Основна мета тестування White Box робиться для перевірки якості коду
Основа для тестових випадків	Тестування може розпочатись після підготовки документа з технічними вимогами	Тестування може розпочатися після підготовки робочого документа
Перевірено	Виконується кінцевим користувачем, розробником та тестером	Зазвичай робиться тестером та розробниками
Зернистість	Зернистість низька	Зернистість висока
Метод тестування	Він заснований на методі спроб та помилок	Область даних та внутрішні межі можуть бути перевірені
Час	Це менш вичерпний і трудомісткий процес.	Вичерпний та трудомісткий метод

Продовження таблиці 4.1

1	2	3
Алгоритм тесту	Не найкращий метод для тестування алгоритмів	Найкраще підходить для тестування алгоритмів
Доступ до коду	Доступ до коду не потрібний для тестування чорної скриньки	Тестування білої скриньки вимагає доступу до коду. Таким чином код може бути вкрадений, якщо тестування виконується на стороні
Вигода	Добры підходить і ефективний для великих сегментів коду	Це дозволяє видалити зайві рядки коду, які можуть призвести до прихованих дефектів
Рівень майстерності	Низькокваліфіковані тестувальники можуть тестувати програму, не знаючи про реалізацію мови програмування або операційної системи	Потрібний досвідчений тестувальник з величезним досвідом для тестування білої скриньки
Методи	Еквівалентне розбиття – це методика тестування чорної скриньки. Поділ за еквівалентністю поділяє вхідні значення на допустимі та недійсні розділи та вибирає відповідні значення з кожного розділу тестових даних. Аналіз граничних значень перевіряє межі для вхідних значень.	Покриття операторів, покриття філії та покриття шляху – це метод тестування Білої скриньки. Оператор Coverage перевіряє, чи кожен рядок коду виконується хоча б один раз. Покриття гілки перевіряє, чи кожна гілка виконується хоча б один раз. Метод покриття шляху перевіряє усі шляхи програми.

Закінчення таблиці 4.1

1	2	3
Недоліки	Оновлення сценарію тестування автоматизації необхідно, якщо ви часто змінюєте програму	Автоматизовані тестові випадки можуть стати марними, якщо кодова база швидко змінюється.

4.2 Тестування програми

Метод функціональних діаграм або діаграм причинно-наслідкових зв'язків допомагає систематично вибирати високорезультативні тести. Крім цього, метод функціональних діаграм дає корисний побічний ефект, оскільки дозволяє виявляти неповноту та неоднозначність вихідних специфікацій.

Функціональна діаграма – це формальна мова, якою транслюється специфікація, написана природною мовою.

Методика використання функціональних діаграм:

– специфікація розбивається на "робочі" ділянки, тому що для великих специфікацій функціональні діаграми стають надто громіздкими. Наприклад, під час тестування компілятора як робочої ділянки можна розглядати кожен окремий оператор мови програмування;

– у специфікації визначаються причини та наслідки. Причина – це окрема умова введення або клас еквівалентних вхідних умов. Наслідок – це вихідна умова (результат виконання програми). Наприклад, якщо при виконанні програми оновлюється вміст деякого файлу, то зміна в ньому є результатом виконання програми, а повідомлення, що підтверджує, – вихідною умовою;

– причини та наслідки визначаються шляхом послідовного читання специфікації. Кожній причині та наслідку приписується унікальний номер;

– аналізується семантичний зміст специфікації, яка перетворюється на булевський граф (функціональну діаграму), що пов'язує причини та наслідк;

– діаграма доповнюється примітками, що задають обмеження та описують комбінації причин та (або) наслідків, які є неможливими через синтаксичні або зовнішні обмеження;

– шляхом методичного простеження станів умов діаграми вона перетворюється на таблицю рішень з обмеженими входами. Кожен стовпець таблиці рішень відповідає тесту;

– стовпці таблиці рішень перетворюються на тести.

Процедура генерації таблиці рішень полягає в наступному:

– вибрати деяке слідство, яке має бути 1;

– знайти всі комбінації причин (з урахуванням обмежень), які встановлять це слідство в 1, прокладаючи з цього слідства зворотну трасу через діаграму;

– побудувати стовпець у таблиці рішень для кожної комбінації причин;

– для кожної комбінації причин визначити стан всіх інших наслідків і помістити їх у відповідний стовпець таблиці рішень.

Таблиця 4.2 – Причини та наслідки

Причини та наслідки							
1	2	3	4	5	6	7	8
Причини та наслідки	У/п	Значення	Помітки про присутність				
			1	2	3	4	5
Причини	П [1]	файли однакового формату	*				
	П [2]	один з файлів не має тексту		*			
	П [3]	обидва файли містять текст			*		
	П [4]	користувач не вибрав перший файл				*	
	П [5]	користувач не вибрав другий файл					*
Наслідки	Н [1]	виконання зіставлення двох файлів	*				
	Н [3]	Виведення повідомлення, що файл пустий		*			
	Н [4]	виконання зіставлення двох файлів			*		

Закінчення таблиці 4.2

1	2	3	4	5	6	7	8
	Н [5]	виведення повідомлення про те, шо користувач не вибрав файл				*	
	Н [6]	виведення повідомлення про те, шо користувач не вибрав файл					*

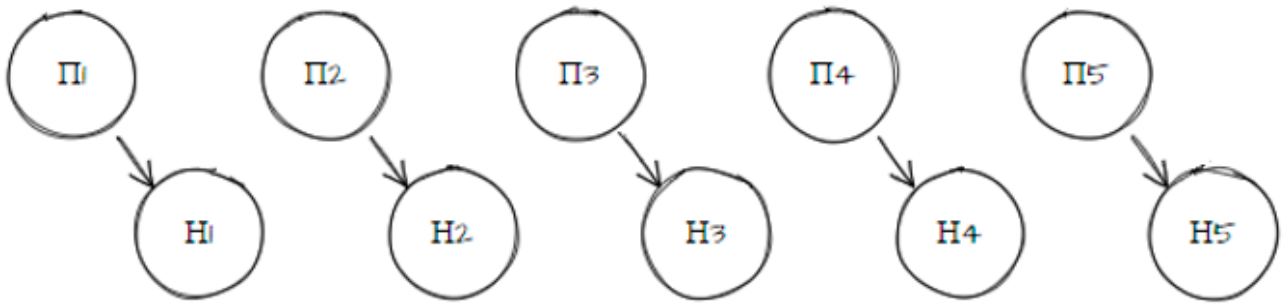


Рисунок 4.1 – Причини та наслідки

4.3 Налагодження

Налагодження – це процес пошуку та видалення помилок із програмного забезпечення. Помилки виникають у програмах, коли рядок коду або інструкція конфліктує з іншими елементами коду. Ми також називаємо помилки чи дефекти апаратними помилками. Суть у тому, що програма не працює належним чином через помилку.

Процес налагодження:

– відтворити ситуацію помилки;

– описати помилку. Отримати якомога більше інформації від користувача, щоб визначити точну причину;

– записати стан програми при появі помилки. Намагатися отримати всі значення змінних та стану програми в цей час;

– проаналізувати статус на основі змінних та дій на даний момент. Виходячи з цього, шукати причину помилки;

– виправити повідомлену помилку, але також перевірити, чи не було введено нову помилку.

Щоб налагодити програму, програміст повинен почати з виявленої помилки. Потім він повинен ізолювати вихідний код з помилкою та виправити помилку. Програміст мусить знати, як виправити помилку. Це вимагає знання аналізу та усунення проблем. Коли помилку усунуто, програмне забезпечення готове до використання.

Протокол налагодження програми представлено в таблиці 4.3

Таблиця 4.3 – Протокол налагодження програми

Опис помилки	Опис ситуації	Способи усунення	Дії, що були застосовані для усунення
Строка, яка була змінена, а не додана фарбуэться у зелений (колір доданого)	Виникає при зіставленні будь-яких файлів	Написати алгоритм, який буде викрашевати одну строку у червоне, якщо вона відрязняється від строки у попередньому файлі	Фарбування строки у червоний, якщо вона була змінена
Збереження файлу відбувається до папки проєкту	Виникає при збереженні файлу користувачем	Реалізувати можливість користувача вибирати директорію, до якої буде зберігатися файл	Реалізована можливість збереження файлу до вибраної директорії
Збереження тільки другого файлу	Виникає при збереженні файлу користувачем	Реалізувати можливість користувача зберігати будь-який з двох вибраних файлів за вибором.	Реалізована можливість збереження файлів за вибором користувача

АНАЛІЗ ТА ВИСНОВКИ

Актуальність інтелектуальних текстових редакторів, а також глобальність їх використання ставить задачі покращення початкового продукту, що і було взято за основу дипломної роботи.

Фокус роботи полягав у розробці додатку, який в подальшому зможе полегшити використання інтелектуальних текстових редакторів, а також розширить можливості роботи з ними.

Задля досягнення мети було проведено опитування користувачів інтелектуальних текстових редакторів з метою визначення функцій, які вони хотіли б бачити в інтелектуальних текстових редакторах.

В ході ретельного аналізу потреб цільової аудиторії була визначена основна – організація можливості порівняння документів, це і було поставлено головною ціллю дипломної роботи. В ході аналізу наявних популярних текстових редакторів було виявлено, шо багато з них мають відкритий код, що дає можливість використання нових рішень та плагінів.

Спираючись на опитування були розроблені вимоги для програмного застосунку, а також виконано його проєктування і безпосередньо розробка.

Мета роботи була досягнута за допомогою розробки додатку, який полегшує роботу користувача та дозволяє зіставляти різні версії документів безпосередньо в одному додатку.

Експлуатаційне призначення було досягнуто створенням програмного застосунку, який полегшує користувачу роботу з версіями документу. Програмний застосунок дозволяє користувачу бачити зміни у документах, бачити додані, чи видалені фрагменти та змінювати документи за бажанням, задля отримання нового кінцевого документу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Стаття з Web-сайту : Habr.com. SimbirSoft: Методи збирання вимог або «Як зрозуміти, що хоче замовник?» – 2016 – 16 серпня – Режим доступу : URL : https://habr.com/ru/company/simbirsoft/blog/307844/ – Дата звернення 25 лютого 2022 р.
2. Стаття з Web-сайту : Stackoverflow.com – Integrated development environment – 2021 Режим доступу : URL : https://insights.stackoverflow.com/survey/2021#technology-most-popular-technologies – Дата звернення 15 лютого 2022.
3. Чернев Д. А. Технологія програмування: Підручник для коледжів. /– Т.: «Ўқитувчи», 2003.—240 с.
4. Брусенцова Т. П. Проектування інтерфейсів користувача: посібник для студентів спеціальності 1-47 01 02 «Дизайн електронних та веб-видань» / Т. П. Брусенцова, Т. В. Кішкурно. – Мінск : БГТУ, 2019. – 172 с.
5. Купер А. Про інтерфейс. Основи проектування взаємодії. /А. Купер, Д. Кронин, Р. Рейман / Пер. с англ. – Символ-Плюс, 2009 – 681 с.
6. Тидвелл Д. Розробка інтерфейсів користувача. / Пер. с англ – «Пітєр», 2008. – 416 с.
7. Головач В. Дизайн інтерфейсу. Мистецтво мити слона. http://www.usethics.ru, Версія 2.09, оновлення від 31.8.2009. – 94 с.
8. Кармайкл Е. Швидка та якісна розробка програмного забезпечення. / Е. Кармайкл, Д. Хейвудс. / Пер. с англ – «Вільямс». 2003. – 400 с.
9. Константайн Л. Разработка программного обеспечения / Л. Константайн, Л. Локвуд / Пер. с англ – «Пітєр». – 2004 р. – 592 с.
10. Макконнелл С. Ідеальний код. Майстер клас / Пер. с англ. — М. : «Російська редакція», 2010. — 896 с.

11. Стаття з Web-сайту : Docs.microsoft.com – OpenFileDialog Класс – Режим доступу: URL:https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.openfiledialog?view=windowsdesktop-6.0 – Дата звернення 30 травня 2022 р.
12. Бейзер Б. Тестування чорної скриньки. Технології функціонального тестування програмного забезпечення та систем. – «Пітєр». – 2004 р. – 320 с
13. Стив К. Не примушуйте мене думати. Веб-юзабіліті та здоровий глузд. 3-тє видання. – «Ексмо» – 2021 р. – 256 с.
14. Інженерія програмного забезпечення [Текст] : Навчальний посібник для підготовки кваліфікаційної роботи на здобуття ОС Бакалавр/ В. М. Горячкін, О. В. Горбова, О. С. Куроп’ятник; Український державний університет науки і технологій. – Дніпро, 2022. – 137 с

Додаток А

ВИМОГИ ДО ПРОГРАМНОГО ЗАСТОСУНКУ

Розширення для інтелектуального текстового редактора Visual Studio Code «Зіставлення різних версій документу»

Розширення буде використовуватись для зручного порівняння різних версій документа, а також буде мати наступні функції:

- новий документ;
- відмінності.

Функція «новий документ» буде надавати змогу збереження нового документу на основі двох попередніх.

Функція «відмінності» буде підсвічувати відмінностей у строках документую.

Програмний продукт цікавим своєю простотою та широким функціоналом.

Програмний продукт призначений для користувачів, які володіють базовими навичками написання коду.

ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є навчальний план для спеціальності 121 «Інженерія програмного забезпечення» затверджений ректором Українського державного університету науки і технологій 30.06.2018. Та наказ ректора Українського державного університету науки і технологій професора Пшінька О.М. «Про призначення наукових керівників та затвердження тем дипломних проектів» за спеціальністю 121 «Інженерія програмного забезпечення» факультету «Комп’ютерних технологій і систем» по кафедрі «Комп’ютерні інформаційні технології» від 2021р.

ПРИЗНАЧЕННЯ РОЗРОБКИ

Функціональне призначення – програмний продукт використовує різні версії документів користувача, або різних користувачів та полегшує формування нового документу на основі попередніх.

Експлуатаційне призначення – за допомогою розширення виконується зіставлення різних версій документів задля швидкого пошуку відмінностей та створення нового документу за бажанням користувача.

ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

1. Вимоги до функціональних характеристик

Програма повинна забезпечити можливість керування документами, а саме:

- видалення непотрібних фрагментів коду з документу;
- додавання нових фрагментів коду до документу;
- вибір кінцевої версії документу.

До неї належать такі основні сутності:

- вхідні документи;
- зміни в документах;
- вихідний документ.

В програмі необхідно реалізувати формування вихідного документу з урахуванням всіх змін та редагувань вхідних документів користувачем.

Вхідні данні:

- перший вхідний документ;
- другий вхідний документ;
- зміни користувача.

Розширення вхідних документів повинні бути однаковими.

Вихідні дані − кінцевий вихідний документ

В ході роботи програми створюється файл, з таким же разширенням, як і два попередні.

2. Вимоги до надійності

Вимоги до надійності наступні:

- при оновленні даних забезпечити показ відповідних повідомлень про стан роботи програми та результати виконання операцій;

- наявність архівної копії тексту програми на зовнішньому носії;
- наявність резервної копії двох попередніх документів на зовнішньому носії.

3. Вимоги до складу та параметрів технічних засобів

Розроблюваний програмний продукт повинен використовуватись на ЕОМ, що мають:

- 4 ГБ оперативної пам’яті;
- 125 ГБ простору на жорсткому диску;
- процесор Intel Core i5-10400F 2.9(4.3)GHz;
- клавіатуру;
- процесор;
- мишу;
- монітор;
- USB-порт.

4. Вимоги до інформаційної та програмної сумісності

Програмний продукт розробляється для всіх видів операційних систем сімейства “Windows 10” та наступні версії.

Обов’язковим до встановлення є бібліотека “.Net framework 4.5” або вище. Середовище розробки: “Visual Studio 2018 professional”. Система керування базами даних “MS SQL Server 2012” [1, 2].

5. Вимоги до маркування і упаковки

Упаковка програмного продукту, включаючи документацію повинна бути захищена від пошкоджень різного роду (механічних, кліматичних). На упаковці повинно бути вказана назва продукту, номер версії (якщо вона змінювалась), мінімальні системні вимоги. На зворотній стороні упаковки вказується розробник та його юридична адреса.

6. Вимоги до транспортування і зберігання

Транспортування повинне забезпечувати збереження програмного продукту, його цілісність і запобігання несанкціонованого доступу до нього. Програмний виріб міститься на хмарному носії, переданий на флешці.

ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

До складу програмної документації мають входити:

- специфікація;
- текст програми;
- опис програми.

Вся документація до програмного додатку повинна задовольняти вимоги до програмної документації.

Додаток Б

ТЕКСТ ПРОГРАМИ

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using System.Windows.Forms;
using System.Text.RegularExpressions;

namespace WindowsFormsApp2
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }
      private void Button1_Click(object sender,
EventArgs e)
      {
         Download download = new Download();
         Methods method = new Methods();
         string[] arrString1;
         arrString1 = new string[10000];
         download.open_file(richTextBox1,
arrString1);
         string[] words = { "function ", "if ",
"foreach ", "for ", "else ", "while ", "break ",
"endfor", "endif", "endblock" };
         string[] words1 = { "int ", "double ",
"string ", "bool ", "var ", "let ", "const ", "void ",
"object ", "this", "true", "false" };
         string[] words2 = { "p ", "h ", "img", "div",
"var", "span", "ul", "li", "b ", "a ", "h1 " };
         string[] words3 = { "class ", "id ", "style ",
"src ", "alt ", "href", "width ", "height", "aria ",
"anme ", "type " };
         for (int i = 0; i < words.Length; i++)
         {
            method.syntax_highlight(richTextBox1,
words[i], 148, 0, 211);
            method.syntax_highlight(richTextBox1,
words1[i], 0, 0, 255);

         }
         for (int i = 0; i < words2.Length; i++)
         {
            method.syntax_highlight(richTextBox1,
words2[i], 255, 69, 0);
            method.syntax_highlight(richTextBox1,
words3[i], 0, 191, 255);
         }
      }

      private void Form1_Load(object sender,
EventArgs e)
      {
         this.WindowState =
FormWindowState.Maximized;
         this.TopMost = true;
      }

      private void Button2_Click(object sender,
EventArgs e)
      {
         Download download = new Download();
         if (checkBox1.Checked &&
checkBox2.Checked)
         {
            MessageBox.Show("Ви не можете
завантажити файли одночасно, завантажте,
буль ласка, файли окремо");
         } else
         if (checkBox1.Checked)
         {
            download.save_file(richTextBox1);
         } else
         if (checkBox2.Checked)
         {
            download.save_file(richTextBox2);
         } else
         if (!checkBox1.Checked &&
!checkBox2.Checked)
         {
```

```
            MessageBox.Show("Ви не вибрали файл для завантаження");
        }

    }

    private void Button3_Click(object sender, EventArgs e)
    {
        Methods method = new Methods();
        if (richTextBox1.Lines.Length == 0 || richTextBox2.Lines.Length == 0)
        {
            MessageBox.Show("Один, або обидва файла не завантажені, завантажте будь ласка файли");
        } else
        {
            method.algorithm(richTextBox1, richTextBox2);
        }

    }

    private void Button4_Click(object sender, EventArgs e)
    {
        string[] arrString2 = new string[10000];
        Download download = new Download();
        Methods method = new Methods();
        download.open_file(richTextBox2, arrString2);
        string[] words = { "function ", "if ", "foreach ", "for ", "else ", "while ", "break ", "endfor", "endif", "endblock" };
        string[] words1 = { "int ", "double ", "string ", "bool ", "var ", "let ", "const ", "void ", "object ", "this", "true", "false" };
        string[] words2 = { "p ", "h ", "img", "div", "var", "span", "ul", "li", "b ", "a ", "h1 " };
        string[] words3 = { "class ", "id ", "style ", "src ", "alt ", "href", "width ", "height", "aria ", "anme ", "type " };
        for (int i = 0; i < words.Length; i++)
        {
            method.syntax_highlight(richTextBox2, words[i], 148, 0, 211);
            method.syntax_highlight(richTextBox2, words1[i], 0, 0, 255);
        }
        for (int i = 0; i < words2.Length; i++)
        {
            method.syntax_highlight(richTextBox2, words2[i], 255, 69, 0);
            method.syntax_highlight(richTextBox2, words3[i], 0, 191, 255);
        }

    }
}
public partial class Download
{
    public void open_file(RichTextBox rtb, string[] arr_name)
    {
        var filePath = string.Empty;
        using (OpenFileDialog openFileDialog = new OpenFileDialog())
        {
            openFileDialog.InitialDirectory = "c:\\Desktop";
            openFileDialog.Filter = "txt files (*.txt)|*.txt|All files (*.*)|*.*";
            openFileDialog.FilterIndex = 2;
            int l = 0;
            openFileDialog.RestoreDirectory = true;

            if (openFileDialog.ShowDialog() == DialogResult.OK)
            {
                filePath = openFileDialog.FileName;
                var fileStream = openFileDialog.OpenFile();

                using (StreamReader reader = new StreamReader(fileStream))
                {
                    while (!reader.EndOfStream)
                    {
                        string s1 = reader.ReadLine();
                        arr_name[l] = s1;
                        l++;
                        rtb.AppendText(s1 + '\n');
                    }
                    if (l == 0)
```

```
                {
                    MessageBox.Show("Будь ласка, оберіть інший файл");
                    open_file(rtb, arr_name);
                }
            }
        }

    }
}
public void save_file(RichTextBox rtb)
{
    SaveFileDialog saveFile1 = new SaveFileDialog();
    saveFile1.DefaultExt = "*txt";
    saveFile1.Filter = "TXT Files|*.txt";
    if (saveFile1.ShowDialog() == System.Windows.Forms.DialogResult.OK &&
        saveFile1.FileName.Length > 0)
    {
        rtb.SaveFile(saveFile1.FileName, RichTextBoxStreamType.PlainText);
    }
}
}
public partial class Methods
{
    public void syntax_highlight(RichTextBox rtb, string str, int r, int g, int b)
    {

        Color color = Color.FromArgb(r, g, b);
        int i = 0;
        while (i <= rtb.Text.Length - str.Length)
        {
            i = rtb.Text.IndexOf(str, i);
            if (i < 0) break;
            rtb.SelectionStart = i;
            rtb.SelectionLength = str.Length;
            rtb.SelectionColor = color;
            i += str.Length;
        }
    }

    public void algorithm (RichTextBox rtb, RichTextBox rtb2)
    {
        string[] arrString1, arrString2;
        arrString1 = new string[10000];
        arrString2 = new string[10000];
        for (int i = 0; i < rtb.Lines.Length - 1; i++)
        {
            arrString1[i] = rtb.Lines[i];
        }
        for (int i = 0; i < rtb2.Lines.Length - 1; i++)
        {
            arrString2[i] = rtb2.Lines[i];
        }
        int x = 0;
        for (int i = 0; i < rtb.Lines.Length - 1; i++)
        {
            string str = arrString1[i];

            if (String.Compare(arrString1[i], arrString2[i]) != 0)
            {

                for (int j = 1; j < rtb.Lines.Length - 1; j++)
                {
                    if (str != arrString2[i])
                    {
                        rtb2.Find(arrString2[i], RichTextBoxFinds.MatchCase);
                        rtb2.SelectionColor = Color.Green;
                        x++;
                    }
                    break;

                }
                if (x == 1)
                {
                    rtb2.Find(arrString2[i], RichTextBoxFinds.MatchCase);
                    rtb2.SelectionColor = Color.Red;
                }

            }
        }
    }
}
```