Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: «Розробка веб-застосунку для аналізу якості програмного коду в командних проєктах»

Виконав: студент IV курсу, групи СПс-41
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва спеціальності)

Кузьмак Є.В.
(підпис) (прізвище та ініціали)

Керівник Цуприк Г.Б.
(підпис) (прізвище та ініціали)

Нормоконтроль Стоянов Ю.М.
(підпис) (прізвище та ініціали)

Завідувач кафедри Петрик М.Р.
(підпис) (прізвище та ініціали)

Рецензент Стадник М.А.
(підпис) (прізвище та ініціали)

Тернопіль 2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Петрик М.Р.
(підпис) (прізвище та ініціали)
«_____» _____ 2026 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)

за спеціальністю 121 – Інженерія програмного забезпечення
(шифр і назва спеціальності)

студенту Кузьмаку Євгену Васильовичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Розробка веб-застосунку для аналізу якості програмного коду в командних проєктах»

Керівник роботи Цуприк Галина Богданівн, к.т.н., доц.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « 6 » квітня 2026 року № 4/9-172

2. Термін подання студентом завершеної роботи «22» червня 2026 р.

3. Вихідні дані до роботи: Завдання на розробку вебзастосунку

4. Зміст роботи (перелік питань, які потрібно розробити)
Обрати напрям дослідження та об'єкт, сформулювати, погодити та затвердити тему кваліфікаційної роботи; розробити та затвердити завдання;
проаналізувати завдання, зробити огляд та проаналізувати бібліографічні матеріали щодо стану справ та тенденій в обраному напрямку дослідження;
сформулювати завдання, обґрунтувати цілі дослідження;
розробити та спроєктувати;
описати технології, етапи розробки та сам продукт,
розробити план тестування програмного продукту; апробувати роботу,
оформити допоміжну документацію;
проаналізувати роботу щодо питань з дотримання положень про безпеку життєдіяльності та основи охорони праці; оформити пояснювальну записку; зробити відповідні висновки за результатами виконаної роботи, представити роботу на розгляд ЕК.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Ілюстративна частина кваліфікаційної роботи оформляється у вигляді слайдів, висвітлює основні розділи та етапи роботи та містить:
тему роботи, мету, предмет та об'єкт дослідження,
сформульовані завдання, етапи виконання кваліфікаційної роботи,
короткий аналіз актуальності теми, проєктування та розробка,
технології розробки; тестування та верифікація; результат апробації роботи та висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності,			
основи охорони праці			
Нормоконтроль	Стоянов Ю.М.		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Вибір та погодження теми з керівником.	26.01-01.02.26	виконано
2.	Затвердження теми КР наказом ректора.	06.04.2026	виконано
3.	Формування та затвердження завдання. Аналіз технічного завдання, підбір та аналіз	20.04.2026	виконано
	бібліографічних матеріалів необхідних для виконання		
	кваліфікаційної роботи		
4.	Проєктування та розробка програмного продукту	11.05.2026	виконано
5.	Тестування та дослідна експлуатація програмного	18.05.2026	виконано
	забезпечення		
6.	Створення допоміжної документації. Написання та	25.05.2026	виконано
	перевірка на відповідність вимогам пояснювальної		
	записки		
7.	Формування записки кваліфікаційної роботи	28.05.2026	виконано
8.	Перевірка програмою антиплагіат	.06.2026	виконано
9.	Безпека життєдіяльності, основи охорони праці	.06.2026	виконано
10.	Нормоконтроль	.06.2026	виконано
11.	Попередній захист	09.06.2026	виконано
12.	Захист кваліфікаційної роботи	.06.2026	

Студент _____ Кузьмак Є.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Цуприк Г.Б.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка веб-застосунку для аналізу якості програмного коду в командних проєктах // Кваліфікаційна робота освітнього рівня «Бакалавр» // Євген Васильович Кузьмак // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СПс-41 // Тернопіль, 2026

Ключові слова: програмне рішення; вебзастосунок; об'єктно-орієнтований підхід; аналіз коду; структуровані дані, проєктування; розробка.

Кваліфікаційна робота присвячена розробці програмного забезпечення для аналізу якості програмного коду в командних проєктах.

У першому розділі було проведено аналіз предметної області, що дозволило визначити особливості оцінювання якості програмного коду в умовах командної розробки.

У другому розділі розглянуто ключові аспекти проєктування вебзастосунка, зокрема вибір процесу розробки, побудову архітектури, моделювання структури даних та реалізацію основних компонентів.

В третьому розділі я виконав перевірку працездатності програмного рішення в умовах практичного використання.

Четвертий розділ присвячено безпеці життєдіяльності та основам з охорони праці.

Об'єкт дослідження: процес забезпечення і контролю якості програмного коду, з врахуванням специфіки командної розробки.

Предмет дослідження: методи, засоби, підходи до комплексного аналізу якості програмного коду та їх програмна реалізація.

ABSTRACT

Development of a web application for analyzing the quality of software code in team projects // Qualification work of the educational level «Bachelor» // Yevhen Kuzmak // Ternopil Ivan Pulyu National Technical University, Computer and Information Systems and Software Engineering Faculty, Software Engineering Department, group SPs-41 // Ternopil, 2026

Keywords: software solution; web application; object-oriented approach; code analysis; structured data, design; development.

The qualification work is dedicated to the development of software for analyzing the quality of software code in team projects.

In the first chapter, an analysis of the subject area was conducted, which allowed us to determine the features of assessing the quality of software code in team development.

In the second chapter, key aspects of web application design are considered, in particular, the choice of development process, architecture construction, data structure modeling, and implementation of main components.

In the third chapter, I performed a performance test of the software solution in practical use.

The fourth chapter is devoted to life safety and the basics of occupational health and safety.

Object of research: the process of ensuring and controlling the quality of software code, taking into account the specifics of team development.

Subject of research: methods, tools, approaches to comprehensive analysis of software code quality and their software implementation.

ЗМІСТ

ВСТУП

У процесі розробки програмного забезпечення, з будь-якої предметної області, над яким може працювати як одна людина так і ціла команда чи навіть колектив, традиційно вважається, що основним критерієм для його успішності є правильність роботи коду. Як я розумію, що мається на увазі його придатність виконувати поставлені функції без помилок і відповідно до визначених вимог. Іншими словами, програмний код є правильним, якщо він забезпечує очікуваний результат у передбачених умовах використання.

Однак, на практиці, код, який навіть формально може працювати коректно, не завжди є зручним для подальшого використання. Те ж саме з його модифікацією чи масштабування. Саме в цьому контексті важливим стає поняття якості програмного коду за значно ширшим набором характеристик, а саме його: читабельність, структурованість, узгодженість, рівень складності, придатність до підтримки та інші.

Особливої ваги ці аспекти набувають в проєктах, над якими працює не один, а цілий колектив фахівців і, відповідно, в кожного своє бачення. Тому розробка одного і того ж програмного продукту, який створюється гркпою фахівців може, з високою ймовірністю, призвести до використання різнорідних підходів, стилів кодування та рівня деталізації реалізації. Більше того, навіть за наявності спільних стандартів, забезпечити однакову, стабільну якість коду в усіх частинах проєкту є досить важким завданням.

В результаті, в процесі проєктування швидше за все, на жаль, будуть накопичуватися помилки, пов’язані не з функціональністю, а саме з якістю коду: зростання складності, дублювання логіки, зниження зрозумілості, ускладнення внесення змін тощо. Це, у свою чергу, повпливає на швидкість розробки, стабільність цілої системи, і на ефективність роботи всієї команди.

Отже, виникає задача у створенні програмного рішення, яке дозволить не лише перевіряти правильність "роботи" коду, а й комплексно оцінювати його

якість з урахуванням специфіки командної взаємодії. Саме це і визначає актуальність теми моєї розробки, яка буде в вигляді вебзастосунка.

На відміну від існуючих підходів, більшість із яких орієнтовані на вирішення вузьких задач, наприклад, пошук синтаксичних помилок, я пропоную продукт, який забезпечуватиме комплексний аналіз щодо показників якості програмного коду саме з врахуванням специфіки при роботі команди. Таке рішення дозволить мені отримати цілісне уявлення про стан проєкту і своєчасно виявляти проблемні його місця.

Метою моєї роботи є розробка вебзастосунку для комплексного аналізу якості програмного коду із забезпеченням представлення результатів та врахуванням особливостей командної взаємодії розробників. Тобто, я створюю інструмент, який дозволить "бачити" не лише окремі помилки в коді, а й оцінити загальний стан якості проєкту, створеного силами всієї залученої команди розробників.

Для досягнення поставленої мети мені буде необхідно вирішити наступні задачі:

1. Проаналізувати сучасні підходи, інструменти оцінювання якості програмного коду і визначити їхні обмеження в умовах командної розробки.

2. Дослідити основні показники якості програмного коду (складність, дублювання, читабельність, структурованість, інші).

3. Визначити доцільні застосування метрики.

4. Сформувати вимоги (функціональні та не функціональні) з урахуванням потреб саме командної роботи.

5. Розробити архітектуру програмного рішення, яка забезпечуватиме інтеграцію різних методів аналізу коду та обробку результатів.

6. Реалізувати вебзастосунок для аналізу якості програмного коду з можливістю збору, обробки та представлення даних.

7. Забезпечити підтримку командної взаємодії: відображення частки роботи кожного розробника, змін у коді та динаміки якості.

8. Провести тестування розробленого застосунку.

9. Оцінити ефективність у задачах аналізу якості програмного коду створеного колективом розробників.

Новизна одержаних результатів полягає у розробці підходу до аналізу якості програмного коду. На відміну від існуючих рішень, розглядається якість не як сукупність окремих перевірок, а комплексно (модель-динаміка- команда-інтегральний/комплексний індекс). Сам показник формується вже в процесі командної розробки.

При реалізації проєкту, я прогнозую отримати наступні результати:

-комплексний підхід до оцінювання якості програмного коду. Зокрема об’єднання різнорідних метрик у єдину узгоджену систему з можливістю їх спільної інтерпретації;

-подальший вдосконалення підходу до аналізу за рахунок врахування специфіки роботи команди (розподілу внесення змін між розробниками, частоти модифікацій та повторюваності типових помилок);

-розглядати якість програмного коду як динамічну характеристику, яка змінюється у процесі розробки проєкта; можливість відстеження в часі;

-отримати модель представлення результатів аналізу, яка дозволить поєднати технічні показники якості з інформацією про процес розробки;

-вдосконалити підхід до візуалізації результатів аналізу, орієнтований на швидке виявлення проблемних ділянок коду та закономірностей їх виникнення при командній роботі;

-розробити концепцію комплексного показника якості програмного коду, що дозволить формувати узагальнену оцінку стану програмного проєкту на основі сукупності технічних метрик та параметрів командної розробки.

Роботу заплановано апробувати та опублікувати в матеріалах роботи IX Міжнародної студентської науково-технічної конференції ТНТУ імені Івана Пулюя “Природничі та гуманітарні науки. Актуальні питання”.

1 АНАЛІЗ ВИМОГ ДО ВЕБЗАСТОСУНКУ

Будь-яка програма сама по собі рідко коли створюється. Частіше і важливіше, коли це є рішення для конкретної практичної потреби. Тому перед початком розробки мені було важливо чітко розуміти: які саме проблеми необхідно вирішити, у яких умовах буде використовуватися програмний застосунок та які очікування від неї має користувач.

В моїй роботі, відповідно до вимог до кваліфікаційної роботи [1], я виокремив з предметної області проблему та потребу, а саме підвищення якості програмного коду в командних програмних проєктах [2-4]. В першого погляду, ця задача не є вже й такою новою, адже існує багато інструментів, які дозволяють перевіряти код, знаходити помилки або контролювати дотримання стандартів. Проте на практиці як виявилось використання таких інструментів не завжди дає цілісне розуміння стану проєкту та ситуації в цілому. В більшості випадків, це пояснюється тим, що значна кількість існуючих рішень орієнтовані на аналіз окремих аспектів коду (наприклад, стиль або безпека), працюють із кодом як ізольованим об’єктом, не враховують особливості та специфіку командної розробки [5-7].

Окремо не можу не зазначити, що останнім часом активно розвиваються підходи, пов’язані з автоматизованим або інтелектуальним рев’ю коду. Такі системи можуть генерувати рекомендації або коментарі до змін, однак їх основною метою є підтримка процесу перевірки коду, а не формування узагальненої оцінки його якості. Крім того, подібні рішення часто залежать від складних алгоритмів і не завжди можуть забезпечувати стабільність та однозначність отриманих результатів.

На відміну від існуючих рішень в своїй роботі я основну увагу приділив саме не автоматичній генерації рекомендацій та не загальному оцінюванню якості програмного забезпечення, а саме аналізу якості програмного коду як складової процесу командної розробки. При цьому якість розглядається не як набір окремих

перевірок, а як узагальнена, комплексна характеристика, що формується під впливом змін у коді та взаємодії усіх залучених до роботи розробників.

Отже, в першому розділі я проаналізую предметну область, визначу ключові особливості та специфіку процесу забезпечення якості коду саме в командних проєктах, сформую вимоги до програмного рішення, з врахуванням як технічних, так і організаційних аспектів та особливостей такої розробки [1].

1.1 Аналіз предметної області

Незважаючи на наявність великої кількості сучасних інструментів для аналізу програмного кода, більшість із них зазвичай орієнтовано на виявлення окремих типів помилок чи якихось недоліків і не забезпечують цілісного уявлення про якість кода в межах всього проєкту . До найбільше відомих поширених рішень належать SonarQube, ESLint, Pylint, Checkstyle, PMD, Code Climate, Snyk та DeepSource [8-15]. При чому, повторюсь, що кожен із цих інструментів як правило вирішує лише окремі задачі (контроль стилю, пошук вразливостей, аналіз складності, дублювання коду), а їх використання відокремлено один від одного ускладнює формування узагальненої оцінки якості. Тобто комплексно проблему якості коду не вирішує жоден із наведених інструментів. Крім того, ці інструменти часто не враховують особливостей роботи над проєктом саме команди розробників, яка має певну специфіку. Йде мова зокрема про внесок кожного розробника, характер змін та динаміка розвитку програмного продукту. В результаті контроль якості коду часто може мати фрагментарний характер, не говорити про проєкт як цілісний, і не дозволить повною мірою оцінити стан програмного коду в контексті розробки всього проєкту [16].

Предметна область мого дослідження охоплює процес забезпечення та аналізу якості програмного коду в умовах командної розробки. На відміну від індивідуального програмування, де контроль якості здійснюється однією особою, у командних проєктах цей процес є більш складним і багатофакторним [17-19].

У сучасній розробці програмного забезпечення код постійно змінюється: додається новий функціонал, виправляються помилки, виконується рефакторинг. Кожна така зміна буде впливати на "загальний стан" системи. Не менш важливим є і те, що різні розробники можуть по-різному підходити до вирішення одних і тих самих задач, а це, в свою чергу може призвести до "неоднорідності" кода.

При таких умовах якість програмного коду формується під впливом кількох груп факторів, зокрема сюди я можу віднести технічні характеристики коду (його складність, наявність дублювання, відповідність стандартам, структурованість), динаміку змін (відображає, як часто змінюються окремі частини коду та наскільки стабільною є його структура в цілому), командний фактор (включає внесок різних розробників, узгодженість їх дій та повторюваність типових помилок).

Не можу не відзначити, що важливою особливістю є й те, що навіть якісний, з технічної точки зору, код може бути досить складним у підтримці. Це може статись, якщо він є незрозумілим для інших членів команди або реалізований без урахування загальної структури проєкта.

Традиційно для оцінювання якості коду використовуються різноманітні метрики та інструменти аналізу, які власне і дають змогу виявляти окремі проблеми [21]. Однак, у більшості випадків, результати такого аналізу залишаються доволі загальні і можуть потребувати додаткової інтерпретації.

В окремий напрям я виділю засоби підтримки рев’ю коду, які націлені в основному на організацію взаємодії між розробниками під час перевірки внесених змін. Важливим є те, що вони зосереджені переважно на процесі обговорення, а не на комплексному аналізі якості.

Також варто відмежувати дане дослідження від підходів, що використовують інтелектуальні або мультиагентні системи для автоматичного аналізу коду. Незважаючи на їх перспективність, такі рішення орієнтовані передусім на генерацію рекомендацій і не завжди забезпечують узгоджене представлення результатів у межах усього проєкту. В моїй ж роботі якість програмного коду я розглядаю як комплексну та динамічну характеристику, яка

формується у процесі командної розробки та потребує узагальненого представлення для ефективного аналізу.

Таким чином, аналіз предметної області показує, що існує потреба у створенні програмного рішення, яке б:

1. Поєднувало б різні показники якості коду.
2. Враховувало б специфіку командної роботи.
3. Дозволило б відстежувати зміни якості у час.
4. Забезпечувало б зручне та зрозуміле представлення результатів аналізу.

1.1.1 Аналіз існуючих рішень

На сьогодні існує значна кількість програмних інструментів, основним призначенням яких є аналіз якості програмного кода. Вони відрізняються за підходами, функціональними можливостями та рівнем інтеґрації у процес розробки [21]. Однак, основним їх "недоліком" (пишу в дужках, бо це їх призначення, проте, сьогодні яке вже потребує перегляду) вони вирішують окремі задачі і не забезпечують комплексного підходу до оцінювання якості коду в умовах специфіки командної роботи. Тому, з метою більш детального аналізу, далі більш детально розгляну існуючі рішення за основними категоріями. Зокрема:

1. Інструменти для статичного аналізу кода;
2. Платформи контролю якості та аналітики;
3. Інструменти для підтримки рев’ю коду;
4. Інтелектуальні підходи до аналізу кода.

До найбільш поширених засобів аналізу варта віднести іструменти статичного аналізу, які перевіряють код без його виконання, серед яких я виділю SonarQube, ESLint, Pylint, Checkstyle та PMD [8-12]. Ці інструменти дозволяють: виявляти синтаксичні та логічні помилки, контролювати дотримання стандартів кодування, оцінювати складність коду, оцінювати дублювання кода, виявляти потенційні вразливі місця. Їх головна перевага – це автоматизація процесу

перевірки та можливість інтеграції у середовище розробки або процеси безперервної інтеграції.Однак, такі рішення мають ряд і обмежень: орієнтація на окремі аспекти якості коду,відсутність узагальненого представлення результатів, складність інтерпретації великої кількості повідомлень. обмежене врахування контексту розробки. У результаті розробник отримує цілий ряд різних зауважень, які не завжди дозволяють однознаяно оцінити загальний стан проєкту в цілому.

До категорії "платформи контролю якості та аналітики" належать уже більш комплексні рішення: Code Climate, DeepSource та Snyk [13-15], які дають змогу агрегувати результати аналізу, формувати метрики якості, інтегруватись із системами контролю версій, базова аналітика змін в коді.До переваг таких платформ я б відніс більш зручне представлення результатів та часткова підтримка командної роботи. Однак і ці рішення в основному працюють переважно з технічними показниками, зазвичай не формують єдиного інтегрального уявлення про якість, достатньо обмежено враховують поведінку всієї команди та динаміку внесення змін, часто, з метою повного аналіза, потребують використання ще додаткових інструментів.

Якщо говорити про інструменти підтримки рев'ю коду, то вони складають окрему групу, яка забезпечує перевірку змін у коді в процесі командної роботи. Для прикладу можу назвати вбудовані механізми платформ контролю версій. До їх основних функції варта віднести можливість обговорення змін, коментування коду, погодження внесених правок. В цілому, інструменти такого виду і подібні відіграють важливу роль у забезпеченні якості, однак, вони не виконують комплексного аналізу кода, не формують узагальнених показників якості, сильно залежать від людського фактору. Отже, їх варта віднести до допоміжних, не самодостатніх засобів контролю за якості.

Сучасні тенденції розвитку програмної інженерії включають використання інтелектуальних методів аналізу коду, зокрема систем, що генерують рекомендації або автоматично виконують рев'ю. Такі підходи дають можливість виявити більщ складні закономірності, аналізувати код з врахуванням контексту, автоматизувати частину процесу перевірки. Однак, результати таких систем

можуть бути неоднозначними. Їх ефективність залежить від налаштування та якості моделей, а основний акцент є на генерації рекомендацій, а не на системному аналізі якості кода. Крім цього вони не завжди забезпечують узгоджене представлення результатів про проєкт. Отже, мжу сказати, що ці рішення не повністю відповідають задачі формування цілісної картини якості коду в командній розробці .

Таким чином, аналізуючи проведений аналіз ,роблю висновки про те, що сучасні інструменти та підходи до аналізу програмного коду в цілому мають певні обмеження, зокрема: відсутність комплексного підходу до оцінювання якості, орієнтація на окремі аспекти або етапи аналізу, недостатнє врахування особливостей специфіки командної роботи, відсутність механізмів для узагальнення результатів з отриманням єдиної оцінки, обмежені можливості аналізу динаміки змін у коді [21-25].

Отже, існує потреба у створенні програмного рішення, яке б поєднувало б результати різних методів аналізу, враховувало б особливості та специфіку саме командної розробки, дозволило б оцінювати якість коду в динаміці, забезпечувало б цілісне та зрозуміле представлення результатів [17-19].

1.2 Постановка завдання та цілей

Після розгляду предметної області та аналізу існуючих рішень (підрозділ 1.1.) я зробив відповідні висновки про те, що проблема оцінювання якості програмного коду не зводиться лише до пошуку окремих помилок або перевірки відповідності стандартам. В реальних умовах розробки над реальними комерційними проєктами значно важливішим є розуміння загального стану коду, його змін у часі та того, як саме робота різних розробників впливає на цей стан.

На практиці розробники часто стикаються з ситуацією, коли результати аналізу їм надають вигляді великої кількості зауважень або метрик, які достатньо складно об'єднати та зробити відповідні висновки. Це впливає і на прийняття рішень і не дозволяє швидко визначити, чи покращується якість коду, чи навпаки

погіршується. Особливо це відчутно в командних проєктах, де зміни відбуваються постійно, з врахуванням різних підходів та алгоритмів, а відповідальність за різні частини системи розподілена між кількома учасниками.

Отже, очевидно, що є потреба не просто в інструменті перевірки коду, а у програмному рішенні, яке дозволить:

1 - об’єднувати результати різних видів аналізу;

2 - відображати їх у зрозумілому та узагальненому вигляді;

3 - враховувати контекст командної роботи;

4 - відстежувати зміни якості у процесі розвитку проєкту.

Виходячи з сформульованих потреб, я сформулював основне завдання моєї роботи , яке буде звучати так: розробка вебзастосунка, основним призначенням якого буде саме комплексний аналіз якості програмного коду з врахуванням специфіки та особливостей командної роботи над розробкою.

Для досягнення поставленої мети мені буде необхідно вирішити наступні задачі:

1 -визначити перелік показників, які доцільно використовувати для оцінювання якості програмного коду;

2 -розробити підхід до узгодженого об’єднання показників та їх інтерпретації;

3 -реалізувати механізм збору та обробки даних про стан коду;

4 -забезпечити можливість аналізу змін у часі;

5 -передбачити врахування впливу окремих розробників;

6 -розробити зручний інтерфейс для представлення результатів аналізу;

7 -реалізувати функціональність налаштування параметрів аналізу.

Кінцевою ж ціллю є створення інструменту, який дозволить не лише фіксувати присутні в коді проблеми, а й "бачити" загальну тенденцію змін в його якості. В свою чергу, такий підхід сприятиме більш обґрунтованому прийняттю рішень у процесі розробки.

1.3 Пошук акторів та варіантів використання

Після визначення основних завдань розроблюваного програмного продукту (детально описаного в підрозділі 1.2) далі, наступним кроком, я встановлю користувачів, які будуь взаємодіяти із нею. Крім користувачів, я ще опишу типові ситуації використання. Це дозволить мені перейти від загального опису функціональності до більш конкретного уявлення про практичне застосування розроблюваного вебдодатка.

Оскільки передбачено, що розробка є результатом роботи команди, вважає за доцільне розглядати не лише формальні ролі учасників, а їх функції у процесі аналізу якості програмного коду [17-18]. Отже, далі я виділю і опишу ролі акторів (див. табл. 1.1)

Таблиця 1.1 – Основні актори та їх ролі

Актор	**Роль**
Член команди розробки	користувач, який безпосередньо: - створює програмний код, - змінює програмний код. Програма, для нього, є інструментом оцінювання результатів власної роботи. Це дозволить своєчасно виявляти проблеми, а також аналізувати вплив внесених змін на кінцеву якість коду.
Координатор якості коду	-користувач, який здійснює контроль стану програмного коду на рівні проєкту. Його основна задача – аналіз узагальнених показників якості, виявлення проблемних ділянок та відстеження динаміки змін у процесі розробки.
Менеджер проєкта	користувач, зацікавлений у: - загальному стані проєкту, - ефективності процесу розробки. Система надає йому узагальнену інформацію про якість коду, яку можна буде використати при прийнятті управлінських рішень.
Адміністратор системи	користувач, який є відповідальним за: - налаштування параметрів роботи вебзастосунка, - керування доступом, -забезпечення стабільного функціонування системи.

В результаті, визначення акторів дозволяє мені сформувати перелік основних варіантів використання, що я далі і зроблю. До типових сценаріїв взаємодії користувачів із розроблюваним програмним забезпеченням я відніс:

1 - отримання узагальненої оцінки якості програмного коду;

2 - перегляд динаміки змін показників якості;

3 - аналіз окремих змін у коді;

4 - оцінювання внеску членів команди розробки;

5 - виявлення проблемних ділянок коду;

6 - перегляд детальних показників якості;

7 - налаштування параметрів аналізу.

Отже, я означив акторів та варіанти використання, які відображають основні напрямки взаємодії з системою та формують основу для її подальшого проєктування.

Для наочного представлення взаємодії користувачів із системою далі логічним є побудувати діаграму варіантів використання, яка узагальнює визначені сценарії та відображає зв’язки між акторами і функціональністю системи.

Далі, на рис. 1.1, я представлю спрощену діаграму варіантів використання, яка відображатиме основні сценарії взаємодії користувачів із системою. Такий варіант я виконав свідомо. Метою було підвищити читабельність діаграми та акцентувати увагу на ключових функціональних можливостях системи без перевантаження другорядними технічними зв’язками. Варто зазначити, що при необхідності я зможу деталізувати діаграму шляхом введення додаткових варіантів використання та уточнення зв’язків між ними (зокрема із використанням відношень типу <<include>> або <<extend>>). Проте в межах даної роботи я обрав узагальнене представлення, яке дозволить мені зосередитись на основних сценаріях використання системи.

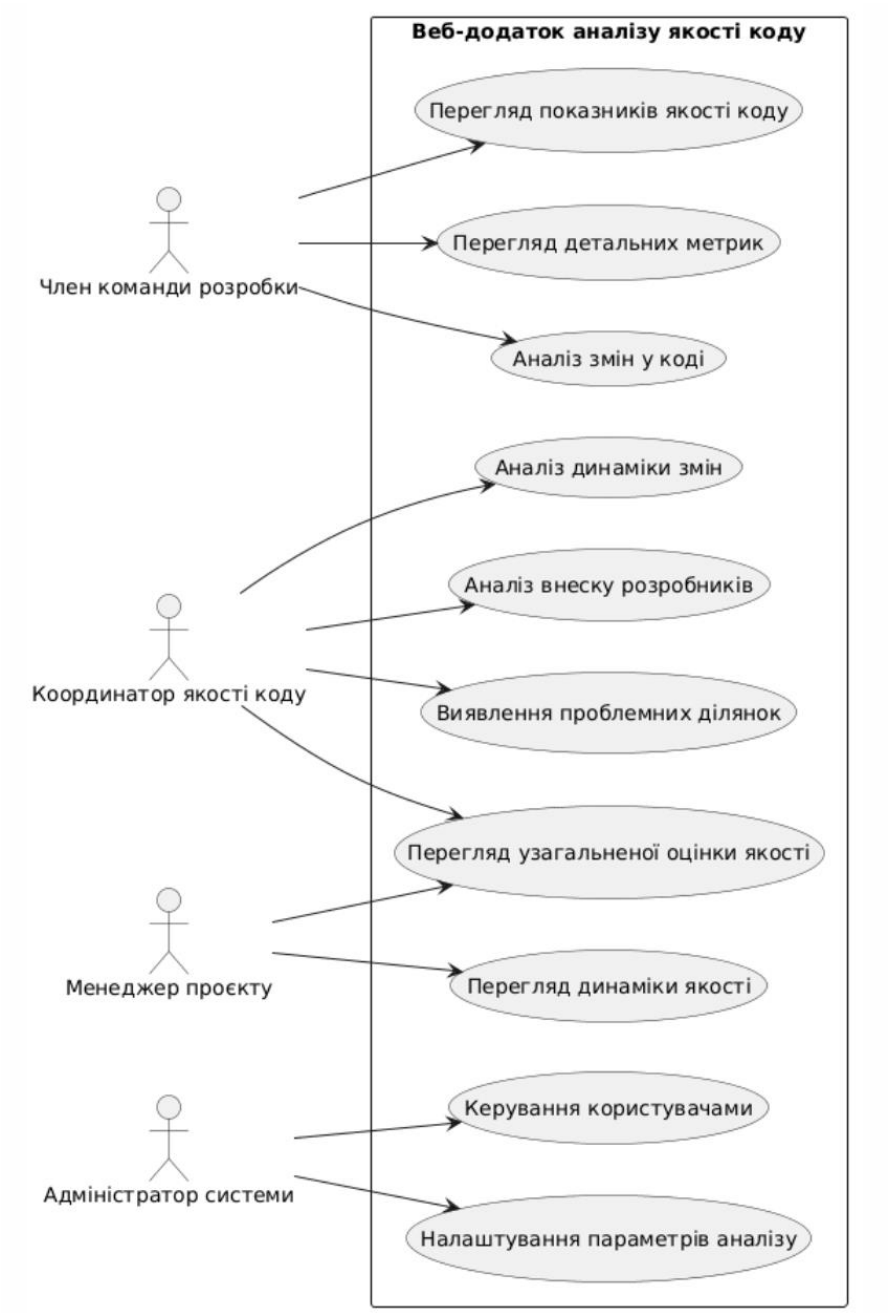

Рисунок 1.1 – Діаграма варіантів використання вебзастосунку

Тут, окрему увагу я приділив підходу до визначення акторів – не за формальними посадами, а відповідно до функцій, які вони виконують у процесі аналізу якості програмного коду. Такий підхід дозволить мені зробити модель більш універсальною та придатною для використання при різних організаційних умовах.

В цілому ж зазначу, що представлена на рисунку 1.1 діаграма варіантів використання в повній мірі відображає основні сценарії взаємодії користувачів із

системою аналізу якості програмного коду та демонструє розподіл функціональності відповідно до ролей користувачів, де кожен актор взаємодіє із системою відповідно до своїх задач у процесі командної розробки. Зокрема, член команди розробки взаємодіє із системою з метою отримання інформації про якість коду та аналізу змін. Координатор якості коду використовує систему для контролю загального стану проєкту та виявлення проблемних ділянок. Менеджер проєкту отримує узагальнені показники якості, необхідні для прийняття управлінських рішень. Адміністратор системи забезпечує налаштування та підтримку її роботи.

1.4 Опис ключових варіантів використання

Після того, як я завершив із визначенням основних акторів та побудував діаграму варіантів використання я більш детально розгляну та опишу ключові сценарії взаємодії користувачів із системою. Це дозволяє конкретизувати її функціональність та зрозуміти, як саме вона найбільш ефективно буде її застосовувати та інтегрувати в процес командної розробки.

У моїй роботі я виокремив п’ять основних варіантів використання, які і відображатимуть типові дії користувачів та найбільш повно характеризуватимуть призначення системи в цілому.

1. Перегляд якості програмного коду.

Цей сценарій є базовим для більшості користувачів системи. Він передбачає отримання узагальненої інформації про стан програмного коду в межах проєкту. Тут користувач має можливість переглянути основні показники якості, які можуть бути представлені у вигляді числових значень, графіків або інтегральної оцінки. Це дозволить швидко оцінити загальний стан коду без необхідності детального аналізу окремих показників.

2. Аналіз змін якості коду.

У межах цього варіанту використання користувач отримує можливість відстежувати зміни якості програмного коду у часі, при цьому система надає

інформацію про динаміку змін. Це дає змогу визначати тенденції розвитку проєкта, оцінювати вплив окремих змін на якість та виявляти моменти погіршення або покращення показників. Цей сценарій є особливо важливим для контролю якості на рівні проєкту.

3. Перегляд метрик якості коду

Цей варіант використання передбачає доступ до детальних показників якості програмного коду. Користувач має можливість аналізувати такі характеристики, як: складність, дублювання, структурованість та інші метрики. Такий аналіз дозволяє виявляти причини зниження якості, аналізувати окремі частини коду, приймати обґрунтовані рішення щодо покращення кода.

4. Аналіз внеску членів команди розробки.

Вебдодаток дає можливість оцінити, вплив кожного окремого учасника на якість програмного коду. У межах цього сценарію користувач може співвідносити зміни в коді з конкретними розробниками, аналізувати повторюваність помилок, виявляти проблемні ділянки, що потребують додаткової уваги. Це сприяє підвищенню ефективності командної взаємодії.

5. Налаштування параметрів програми. Даний варіант використання призначений для адміністратора системи та передбачає можливість зміни параметрів аналізу. Користувач може визначати перелік використовуваних метрик, налаштовувати правила оцінювання якості, керувати доступом до функціональності системи. Таким чином цей сценарій дозволить адаптувати систему під специфіку конкретного проєкту.

У результаті розгляду ключових варіантів використання я конкретизував функціональні можливості системи та визначив основні сценарії її застосування. Це створює основу для формування вимог до програмної системи та подальшого етапу її проєктування.

1.5 Формування вимог до програмного продукту

Далі, враховуючи результати проведеного аналізу предметної області, існуючих рішень та визначених варіантів використання я сформував вимоги до розроблюваного мною вебзастосунку, якими і визначаються його функціональні можливості та основні характеристики, які мені необхідно буде врахувати під час проєктування та вже самої реалізації [27].

Функціональні вимоги визначають перелік задач, які повинна виконувати програма, а саме вона повинна забезпечувати:

1- збір та обробку даних про програмний код;
2- формування показників якості програмного коду;
3- відображення узагальненої оцінки якості;
4- надання доступу до детальних метрик;
5- аналіз змін якості у часі;
6- виявлення проблемних ділянок коду;
7- оцінювання внеску членів команди розробки;
8- підтримку різних ролей користувачів;
9- налаштування параметрів аналізу;
10- керування доступом до системи.

В свою чергу нефункціональні вимоги визначають характеристики системи, які і впливають на якість її роботи. Зокрема, програмне рішення повинне відповідати наступним вимогам:

1- зручність використання (інтерфейс має бути зрозумілим та інтуїтивним).
2- продуктивність (обробка даних повинна здійснюватися без суттєвих затримок).
3- масштабованість (продукт повинен підтримувати роботу з проєктами різного обсягу).
4- надійність (забезпечення стабільної роботи та збереження даних).
5- гнучкість (можливість налаштування під потреби користувачів).
6- безпека (контроль доступу та захист даних).

7- сумісність (інтеграція з іншими інструментами розробки).
8- розширюваність (можливість додавання нових функцій і метрик).

Отже, сформовані вимоги визначають основні характеристики та функціональні можливості розроблюваного вебзастосунку. Це дозволяє перейти до етапу проєктування його архітектури та реалізації.

2 ПРОЄКТУВАННЯ ТА РОЗРОБКА ВЕБЗАСТОСУНКУ

Після етапів визначення вимог до розроблюваної програми та аналізу варіантів її використання, що я детально писав в розділі 1, наступним етапом буде проєктування та реалізація мого програмного рішення. На цьому етапі я сформую внутрішня структура системи, визначу підходи до її розробки, а також оберу інструменти та технології, що забезпечують досягнення поставленої мети.

Таким чином, особливістю розроблюваної програми є необхідність поєднання аналітичної обробки даних [21-24] із зручним представленням результатів для різних категорій користувачів. Це вимагає продуманого підходу до організації архітектури, взаємодії компонентів та реалізації основної логіки системи.

У другому розділі я розгляну ключові аспекти проєктування, зокрема вибір процесу розробки, побудую архітектуру, змоделюю структуру даних та реалізацію основних компонентів [28-30].

2.1 Вибір процесу розробки

Вибором процесу розробки зазвичай визначається загальний підхід до створення програмного застосунку. Саме він впливає на організацію робіт, послідовність їх етапів та взагалі на можливість внесення змін у процесі реалізації. Для розроблюваного вебзастосунка, який орієнтовано на аналіз якості програмного коду, який створюється в процесі командної роботи, коректність та оптимальність цього вибору є особливо важливим, оскільки вимоги до розробки можуть уточнюватися і в процесі її створення.

Серед найбільш поширених підходів до розробки програмного забезпечення такого типу я можу виділити каскадну модель, ітеративну модель та гнучкі методології [31].

Зокрема, при каскадній моделі відбувається послідовне виконання етапів розробки (аналіз вимог, проєктування, реалізація, тестування та впровадження).

До основних переваг такого підходу варта віднести чітку структуру та зрозумілу послідовність дій. Однак є й недоліки, до котрих я віднесу низьку гнучкість. Мається на увазі, що в разі зміни вимог повернення до попередніх етапів є доволі складним та потребує значних витрат часу. Тому для систем, де логіка та підходи до обробки даних можуть і будуть уточнюватись, вибір цього підхіду є не логічним.

В свою чергу гнучкі методології розробки (зокрема підходи, суттю яких є ітеративність та постійний зворотній зв’язок) орієнтовані на швидке створення програмного продукту. Тут також враховується і можливість постійного вдосконалення. Вони добре працюють при командній роботі та для динамічних проєктів. Однак, при їх застосуванні ередбачено наявність повноцінної команди розробників та взаємодія із замовником на постійній основі. Тому для індивідуальної розробки парадигми гнучких методологій до розробки не завжди можуть бути реалізовані у повному обсязі.

Таким чином, проаналізувавши підходи я прийняв рішення в своїй роботі застосувати найбільш підходящий ітеративний підхід, в якому власне і поєднуються переваги структурованості та гнучкості. Саме тут передбачено поділ процесу розробки на окремі ітерації, у межах котрих послідовно виконуються етапи аналізу, проєктування, реалізації та тестування. Я використав ітеративний підхід із елементами Agile, але без повного впровадження методології через індивідуальний характер розробки.

До основних переваг обраного мною ітеративного підходу є:

- можливість поступового уточнення вимог до розроблюваного програмного продукту;

- поетапна реалізація функціональності;

- раннє виявлення помилок і недоліків;

- можливість оцінювання проміжних результатів.

Отже, враховуючи особливості моєї роботи перелічені переваги "означають", що на початкових етапах є можливість реалізувати базову функціональність (наприклад, відображення показників якості коду). Після цього

програмне рішення буде поступово доповнюватись більш складними можливостями, такими як: аналіз динаміки змін або оцінювання внеску членів команди.

Підсумовуючи все вище сказане, роблю висновок ,що вибір ітеративного підходу є оптимальним, оскільки таким чином я забезпечу необхідний рівень гнучкості та матиму змогу адаптувати процес розробки до особливостей задачі. Саме тому це і робить його найбільш доцільним для реалізації програмної системи основною задачею якої є аналіз якості програмного коду.

Окремо пропоную розглянути можливість використання гнучких підходів розробки, зокрема таких, які базуються на принципах методології Agile. Тут передбачено короткі ітерації, постійний зворотний зв’язок та активна взаємодія між учасниками всієї команди. Застосування цієї та подібних методологій є доцільним у тих випадках, коли розробка здійснюється командою та передбачає тісну співпрацю із замовником, оскільки передачено можливість швидкої адаптації до змін вимог та підвищення ефективності саме колективної роботи.

Разом з тим, оскільки в моїй роботі розробка програмного продукту у вигляді вебзастосунку здійснюється як індивідуальний проєкт, це є і обмеженням можливості повноцінного використання усіх елементів гнучких методологій, зокрема регулярних командних взаємодій та формалізованих ролей.

З огляду на це, доцільно є використовувати окремі принципи гнучкого підходу. Зокрема йдеться про ітеративність та поетапне уточнення функціональності без повного впровадження конкретної методології. Це дозволить мені поєднати гнучкість процесу розробки з його керованістю та структурованістю.

2.2 Проектування архітектури вебзастосунка

Проєктування архітектури розробки є одним із ключових етапів, бо саме на цьому рівні і визначається структура програмного рішення, взаємодія його компонентів та підхід до опрацювання даних. Від коректності обрання

архітектури залежить не лише зручність подальшої реалізації, але й можливість розширення функціональності, адаптації до нових вимог та підтримки вебзастосунку в процесі його використання.

Особливістю розроблюваного вебзастосунку є те, що він не обмежується виконанням окремих операцій над кодом, а орієнтований на комплексний аналіз якості програмного коду з урахуванням динаміки змін та специфіки саме командної взаємодії. Це значить те, що архітектура повинна забезпечувати виконання та коректність обробки різнорідних даних, узгодження результатів аналіза, представлення результатів аналізу в зручному для користувача вигляді.

На відміну від багатьох існуючих рішень, де основну увагу зосереджено на інтеграції окремих інструментів аналізу, у моїй роботі акцент я зробив на організації єдиного аналітичного рівня. За його допомогою будуть узагальнюватись результати та який дозволить мені оцінювати якість коду в контексті всього проєкту.

З урахуванням зазначених вимог доцільним є використання багаторівневої архітектури, яка передбачає розподіл системи на окремі логічні рівні.

Архітектуру вебдодатку я умовно поділю на такі основні рівні:

1. Рівень представлення (інтерфейс користувача). Цей рівень "відповідає" за взаємодію користувача із системою. На цьому рівні здійснюється відображення результатів аналізу, візуалізація показників якості та забезпечення доступу до функціональності системи.

2. Рівень прикладної логіки. Тут реалізується основна функціональність системи, в тому числі включно із обробкою запитів користувачів, формування показників якості та управління процесом аналізу.

3. Аналітичний рівень — є ключовим елементом системи. Саме він відповідає за обробку даних про програмний код, узагальнення результатів та формування інтегральних показників якості. Саме цей рівень відрізняє розроблювану систему від існуючих типових рішень.

4. Рівень доступу до даних. Забезпечує збереження та отримання інформації, пов’язаної з програмним кодом, результатами аналізу та користувачами системи.

На відміну від класичних підходів, коли система будується навколо окремих інструментів аналізу, у моїй роботі архітектура орієнтована на централізовану обробку результатів аналізу, що дозволяє формувати узагальнені оцінки якості; підтримку динамічного аналізу, тобто відстеження змін показників у часі; врахування командної взаємодії, зокрема внеску окремих користувачів; гнучкість у додаванні нових метрик та алгоритмів аналізу. Отже, обраний мною підхід дозволить розглядати програмне рішення не як набір окремих інструментів, а як єдине аналітичне середовище.

В загальному вигляді робота вебдодатку відбувається наступним чином: користувач через інтерфейс формує запит на отримання інформації про якість коду, який передається до рівня прикладної логіки. Далі запит обробляється аналітичним рівнем. Тут виконуються необхідні обчислення та формуються результати. Після чого дані передаються до рівня представлення, де вони відображаються у зручному вигляді. Таким чином саме таких підхід є оптимальним і дозволить розділити відповідальність між компонентами системи та забезпечити їх роботу.

Для наочного представлення структури розроблюваного вебзастосунку далі я побудуть архітектурну діаграму, яка відображатиме основні компоненти розробки та взаємозв’язки між ними. Зокрема, архітектуру , яка складається з рівня представлення, прикладної логіки, аналітичного рівня та рівня доступу до даних. Така структура забезпечить логічний розподіл функціональності між компонентами і дозволить реалізувати гнучкий підхід до обробки та аналізу даних.

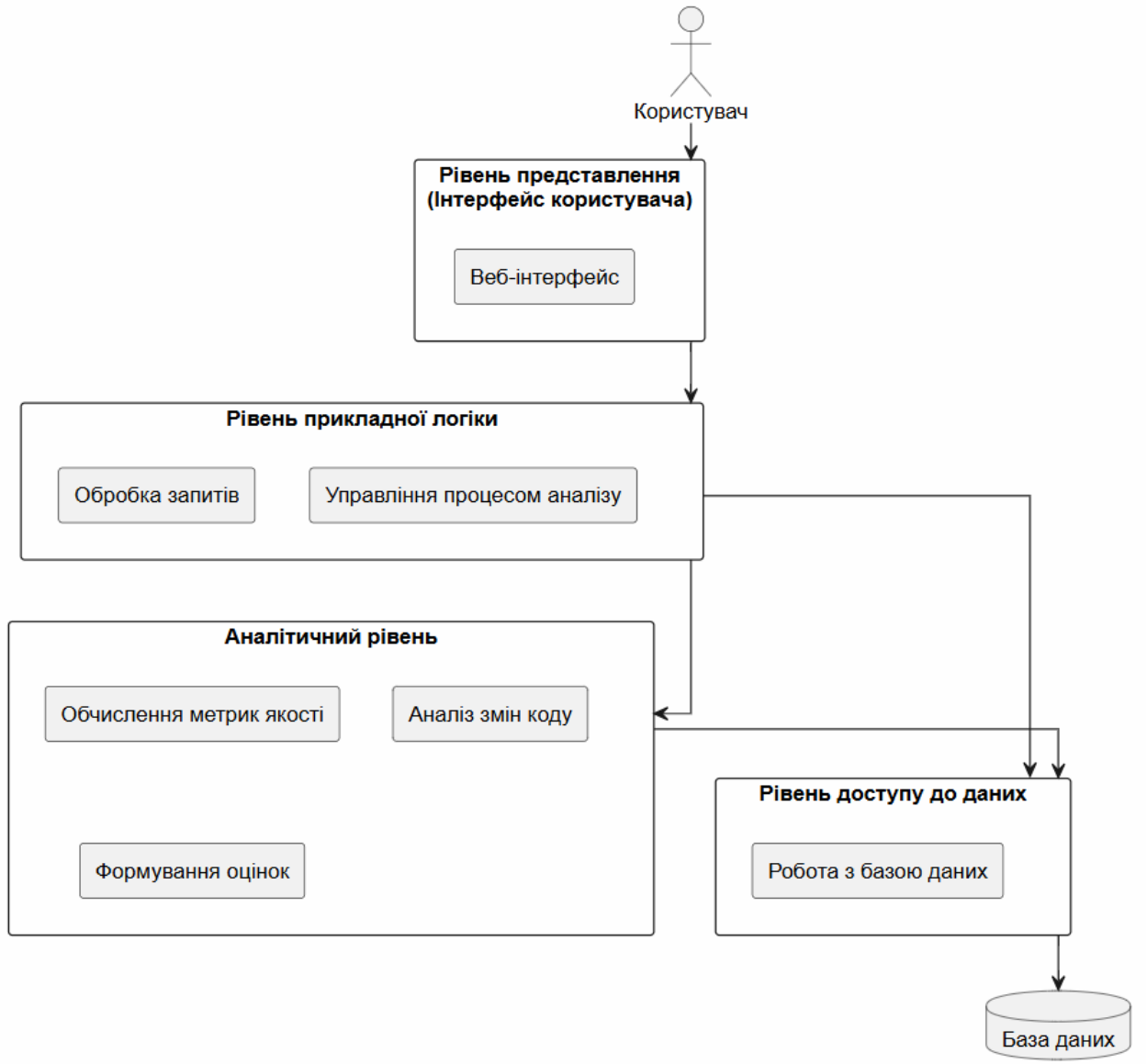

Рисунок 2.1 – Архітектура вебзастосунку аналізу якості програмного коду

На діаграмі 2.1 користувачі представлені узагальнено, оскільки з точки зору внутрішньої структури системи всі ролі взаємодіють із нею через єдиний інтерфейс. Деталізація ролей користувачів наведена на діаграмі варіантів використання в підрозділі 1.3 на рисунку 1.1.

2.3 Побудова схем бази даних

Проєктування безпосередньо бази даних є одним із ключових етапів розробки не лише мого, а й будь-якого якісного програмного продукту [32]. Важливість цього етапу обумовлена тим, що саме на цьому кроці забезпечується

збереження, структуризація та подальша обробка інформації, необхідної для функціонування розроблюваного веб-застосунка.

У моїй роботі база даних повинна підтримувати не лише збереження поточного стану даних, а й можливість аналізу їх змін у часі. Перш за все, це пов’язано з тим, що програму орієнтовано на оцінювання якості програмного коду в динаміці та врахуванням специфіки та особливостей саме командної роботи над проєктом. На відміну від простих інформаційних систем, де фіксується лише кінцевий результат, у моєму випадку база даних виконує ще також і аналітичну функцію, цим самим дозволяючи відстежувати історію змін, зв’язки між розробниками та результати оцінювання коду.

Далі прийшов час побудувати логічну модель, яка визначатиме основні сутності системи та зв’язки між ними без прив’язки до конкретної реалізації СУБД. Для початку, в моєму програмному рішенні я виділив основні сутності, які описав в таблиці 2.1.

Таблиця 2.1 – Основні сутності програми

Сутність	**Опис**
Користувачі	учасники системи, які взаємодіють із веб-застосунком: -розробники, -менеджери, -адміністратори
Проєкти	програмні проєкти, для яких виконується аналіз якості коду
Версії коду	різні стани проєкту у часі
Коміти	конкретні зміни у програмному коді
Метрики якості	показники, за якими здійснюється оцінювання коду
Результати аналізу	значення метрик для конкретних змін у коді
Аналіз проєкту	узагальнена оцінка якості проєкту
Внесок користувачів	інформація про участь розробників у змінах коду

Для наочності я представлю візуалізацію описаної вище структури бази даних, побудувавши логічну модель (див.рис. 2.2), яка демонструватиме основні сутності системи та зв’язки між ними [33].

Далі на рисунку 2.2 я представлю розширену логічну модель бази даних моєї програмної системи, яка відображатиме основні сутності предметної області та зв’язки між ними. Вони забезпечують збереження та обробку інформації про процес аналізу якості програмного коду. Центральною сутністю є «Проєкт», оскільки саме він об’єднує всі елементи аналізу. Крім цього "Проєкт" виступає основною одиницею організації даних. Важливо, що в межах одного проєкту може існувати кілька версій коду, що дозволить відстежувати його розвиток у часі і аналізувати зміни між різними станами.

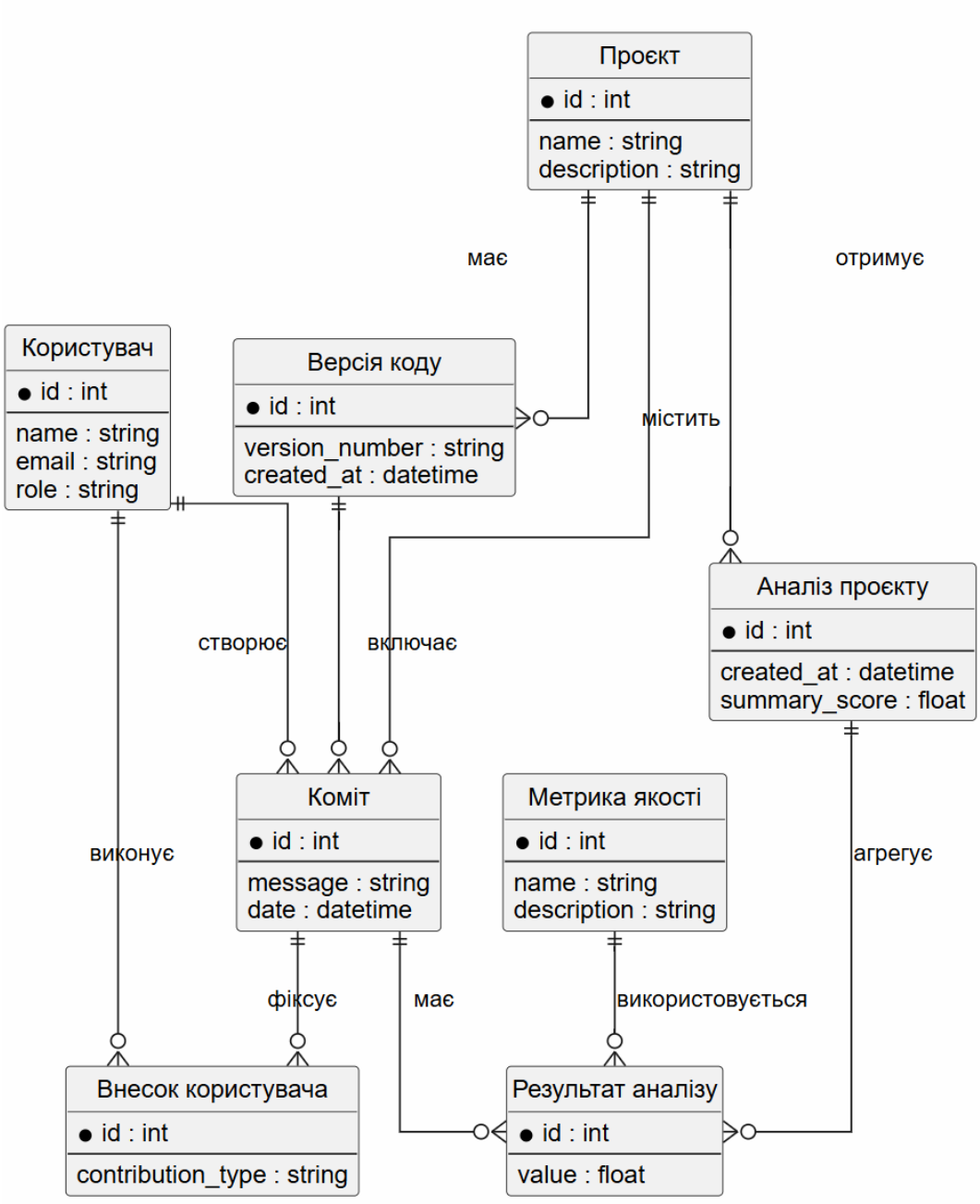

Рисунок 2.2 – ER-діаграма бази даних

Тут, основною одиницею аналізу є «Коміт», яким фіксуються конкретні зміни у програмному коді. Кожен коміт пов’язаний із користувачем. Це дозволить визначати авторів змін та аналізувати внесок кожного члена команди розробки. Результати аналізу якості коду зберігаються у вигляді значень метрик. Це дасть змогу оцінити різні аспекти програмного коду та порівнювати їх між собою. Окреме виділення сутності метрик забезпечить гнучкість системи та можливість розширення критеріїв оцінювання без зміни загальної структури бази даних. Крім цього, я додатково передбачив узагальнення результатів на рівні проєкту. Це дозволить формувати інтегральну оцінку якості та відслідковувати загальний стан програмного продукту в цілому.

Далі я побудував фізичну модель, яка реалізує логічну структуру у вигляді таблиць та їх зв’язків у базі даних. Основні таблиці системи та іх опис я надам в таблиці 2.2.

Таблиця 2.2 – Основні таблиці фізичної моделі системи

Назва	Опис
Користувачі (users) (id, name, email, role)	зберігає інформацію про користувачів системи
Проєкти (projects) (id, name, description)	містить дані про програмні проєкти
Версії коду (versions) (id, project_id, version_number, created_at)	фіксує стани коду у часі
Коміти (commits) (id, project_id, user_id, message, date)	зберігає зміни у коді
Метрики якості (metrics) (id, name, description)	визначає показники оцінювання
Результати аналізу (analysis_results) (id, commit_id, metric_id, value)	містить значення метрик
Аналіз проєкту (project_analysis) (id, project_id, created_at, summary_score)	узагальнена оцінка якості
Внесок користувачів (user_contributions) (id, user_id, commit_id, contribution_type)	відображає участь у змінах коду

Діаграма фізичної моделі бази даних моєї програми матиме вигляд на рис. 2.3.

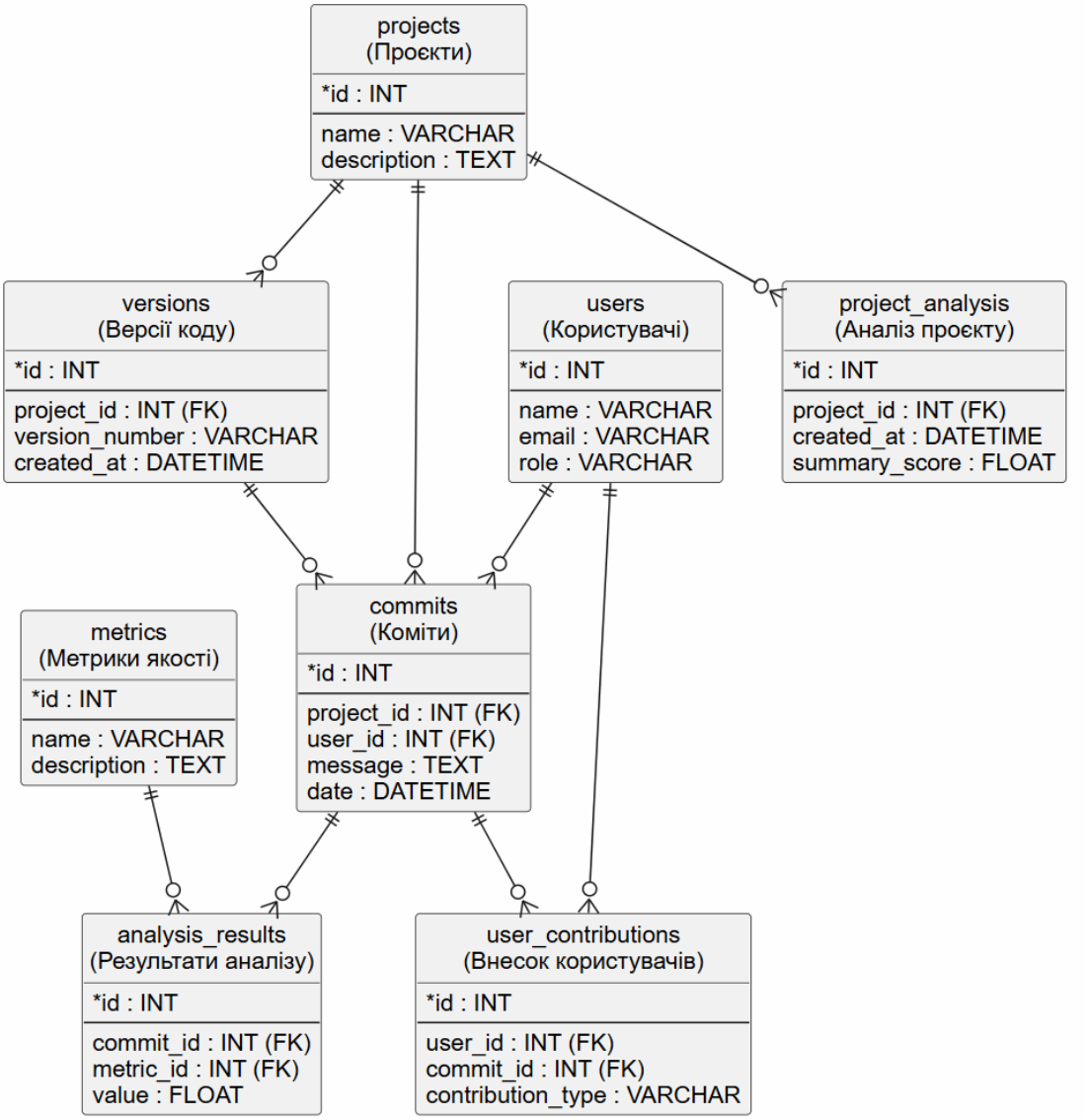

Рисунок 2.3 – Фізична модель бази даних програмної системи

Фізична модель бази даних відображає реалізацію логічної структури у вигляді таблиць, їх полів та зв’язків у системі керування базами даних. За допомогою неї я можу продемонструвати, як саме сутності предметної області трансформуються у реляційну структуру. Тут, зв’язки між таблицями реалізуються за допомогою зовнішніх ключів. Це забезпечує цілісність даних та можливість виконання аналітичних запитів. Така структура підтримує збереження історії змін, аналіз якості коду та оцінювання внеску користувачів у розробку проєкту.

Таким чином, я побудував логічну та фізичну моделі бази даних програмної системи. Запропонована структура забезпечить збереження даних про проєкти, зміни у коді, результати аналізу та внесок користувачів. Також передбачено можливість виконувати аналіз якості програмного коду в динаміці [34].

2.4 Побудова UML-діаграми класів

Після визначення архітектури системи (підрозділ 2.2) та структури бази даних (підрозділ 2.3) наступним етапом буде проєктування внутрішньої структури програмного забезпечення. Для цього використовується UML-діаграма класів, яка дозволить мені описати основні компоненти системи, їх властивості, а також взаємозв'язки. Діаграма класів відобразить логіку реалізації моєї програмної системи на рівні об'єктно-орієнтованого підходу. Також вона дасть можливість зрозуміти, як саме будуть організовані дані в коді та яким чином здійснюватиметься взаємодія між окремими частинами системи. оскільки особливістю розроблюваного мною веб-застосунку є наявність аналітичного компонента, який дозволить обробляти результати аналізу якості коду, то при побудові діаграми класів мені важливо було відобразити не лише базові сутності, але й логіку їх взаємодії в процесі аналізу.

Для наочності представлення структури програмної системи та взаємодії її компонентів далі на рисунку 2.4 я представлю побудовану UML-діаграму класів [35, 36].

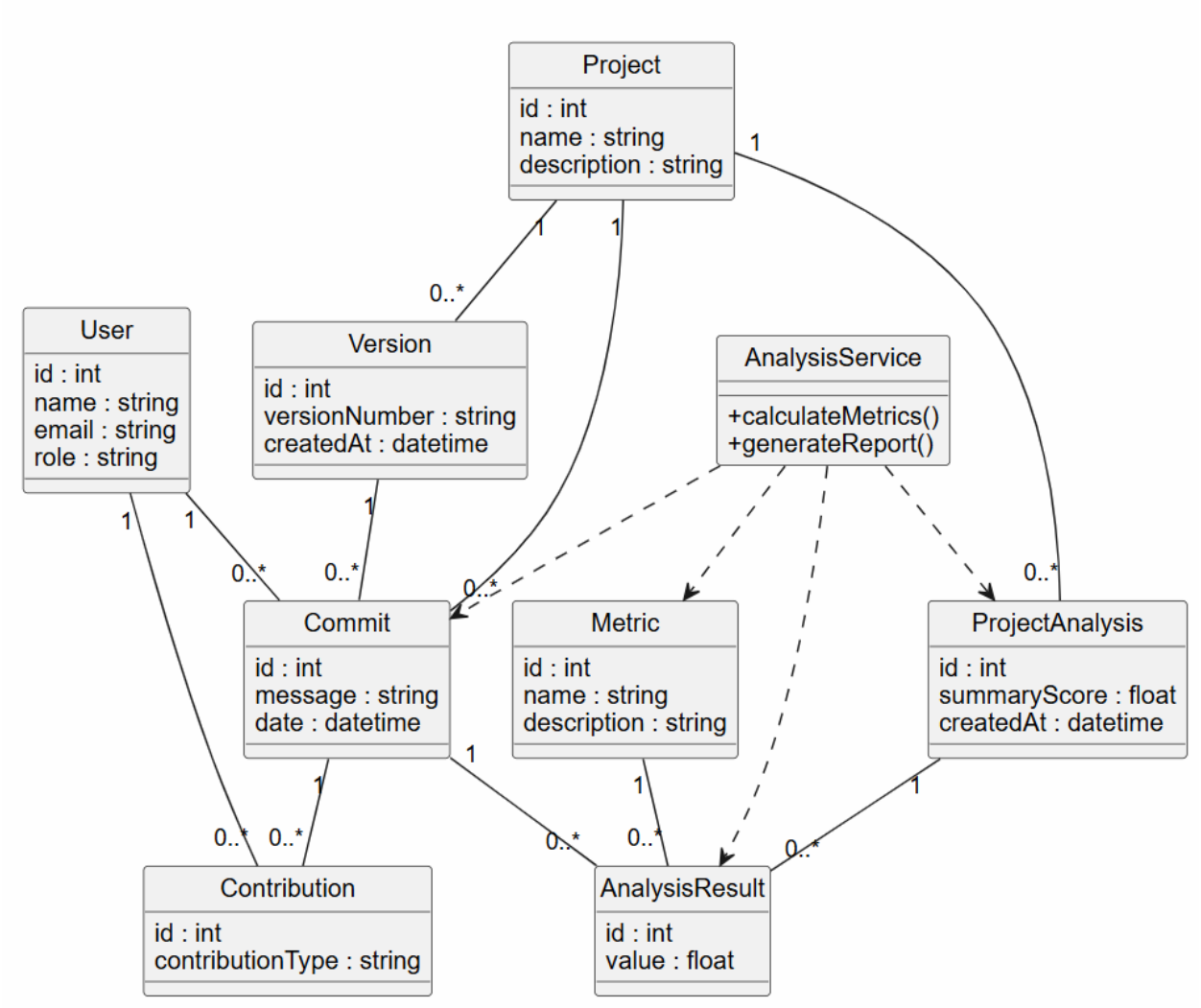

Рисунок 2.4 – UML-діаграма класів програмного забезпечення для аналізу якості програмного коду

На рисунку 2.4 я представив UML-діаграму класів, яка відображає основні компоненти розроблюваної мною програми та взаємозв'язки між ними. Основу системи складають класи, які відповідають сутностям предметної області, йдеться про користувача, проєкт, коміт, метрику якості та результат аналізу. За допомогою цих класів забезпечується збереження та представлення даних у системі. Окрему роль відіграє клас AnalysisService, за допомогою якого я буду реалізувати логіку обробки даних та формувати результатти аналізу. Саме клас AnalysisService відповідає за обчислення метрик якості та узагальнення отриманих результатів. Наявність окремого класу для аналізу проєкту дозволяє

формувати узагальнені показники якості та оцінювати стан програмного продукту в цілому. Зв’язками між класами я відбразив структуру взаємодії компонентів системи. Наприклад, користувач пов’язаний із комітами та внесками, а це дозволяє відстежувати участь розробників у зміні коду. В свою чергу, Commit пов’язані з результатами аналізу, що дає можливість оцінювати якість коду для кожної зміни. Отже в діаграмі класів я відобразив не лише структуру даних, але й логіку функціонування системи в цілому. Це є важливо для подальшої реалізації мого програмного рішення.

Таким чином, побудувавши UML-діаграму класів я визначив структуру мого застосунка та взаємодію його основних компонентів. Запропонована модель дозволить мені реалізувати функціональність аналізу якості програмного коду та забезпечить гнучкість системи та можливість її подальшого розширення.

2.5 Вибір мови та середовища розробки

Під час вибору мови програмування та середовища розробки найбільшу увагу я приділяв не лише популярності технологій, а й їх відповідності конкретним задачам, які ставляться перед розроблюваним програмним рішенням. Оскільки вебзастосунок я орієнтував на аналіз якості програмного коду, для мене важливо було забезпечити ефективну та зручну обробку данних. Так само важливою є можливість узагальнення отриманих результатів та їх зрозуміле представлення користувачу. Також, система повинна бути достатньо гнучкою, оскільки я планую, що в подальшому можна буде розширювати перелік метрик та вносити зміни в логіку аналізу без суттєвого перепроєктування.

У якості основної мови програмування я зупинив свій вибір на JavaScript [37, 38]. Аргументую свій вибір тим, що отримаю можливість використовувати єдиний підхід як для клієнтської, так і для серверної частин системи. Такий вибір є оптимальним, бо знижує складність розробки, оскільки не потребує переходу між різними мовами програмування. Також це спростить обмін даними між компонентами. Окрім цього, велика кількість доступних

бібліотек і інструментів дозволить мені зосередитись ще й на реалізації логіки аналізу, а не лише на технічних деталях.

Для реалізації серверної частини я використав платформу Node.js [39]. Це рішення добре підходить для обробки великої кількості запитів та роботи з асинхронними операціями. Це є важливо у контексті аналізу програмного коду, коли обробка даних може виконуватися паралельно чи потребувати доступу до різних джерел інформації. Використання Node.js дозволить мені ефективно реалізувати обчислення метрик та формувати узагальнені результати.

Клієнтську частину я реалізовуватиму з використанням бібліотеки React. Вона підтримує компонентний підхід до побудови інтерфейсу. Це дозволить мені структурувати інтерфейс у вигляді незалежних елементів, які легко можна буде змінювати та повторно використовувати. Такий підхід є оптимальним та доцільним для застосунку в якому результати аналізу зможуть відображатися у різних формах (таких як таблиці, узагальнені показники, порівняння), а також буде можливість частого оновлення без перезавантаження сторінки.

Для збереження даних я скористуюсь можливостями реляційної системи керування базами даних. Цей вибір обумовлений наявністю чітко визначених зв’язків між сутностями, зокрема між проєктами, комітами, результатами аналізу та користувачами. Реляційна модель дозволить мені забезпечити цілісність даних та виконувати складні запити, необхідні для формування аналітичних показників. Крім того, структура такого типу буде добре підходити для задач, в яких важливо відстежувати історію змін і взаємозв’язки між елементами системи.

Середовище розробки я обирав з врахуванням зручності роботи, підтримки обраних мною технологій та наявності інструментів для налагодження програмного кода. Використання ж сучасного інтегрованого середовища дозволить мені ефективно організувати як безпосередньо сам процес розробки, так і працювати з різними компонентами системи, а також швидко виявляти і виправляти помилки.

Отже, підсумовуючи, в цьому підрозділі я обгрунтував вибір стеку технологій, до якого включено мову програмування JavaScript, платформу Node.js

для серверної частини, для реалізації клієнтського інтерфейсу - бібліотеку React, а реляційну систему керування базами даних для збереження інформації [37-39]. Такий вибір в вважаю оптимальним оскільки він відповідає поставленим у роботі завданням тиа з точки зору поєднання гнучкості, зручності розробки та можливості реалізації аналітичної функціональності системи.

2.6 Реалізація основних класів та методів

Після проєктування структури системи та визначення взаємозв'язків між її основними компонентами (підрозділи, 2.2, 2.3, 2.4) наступним кроком я буду реалізувати основні класи та методи, які забезпечуватимуть функціонування мого застосунку. На цьому етапі особливу увагу я приділив не лише збереженню даних, а й організації логіки аналізу якості програмного коду та взаємодії між компонентами системи. Також мене цікавила можливість подальшого розширення функціональності.

Під час реалізації я використав об'єктно-орієнтований підхід. Його парадигми дозволять мені розділити програму на окремі логічні компоненти, що значно спростить підтримку програмного забезпечення, повторне використання коду та внесення змін у майбутньому. Важливо, що кожен клас відповідатиме за окрему частину функціональності. Це дозволить уникнути надмірної залежності між компонентами застосунка. В загальному ж основу системи складають класи, які пов'язані з роботою користувачів, проєктів та аналізу програмного коду. Клас користувача забезпечує збереження інформації про учасників системи та їх ролі. В свою чергу клас проекту відповідає за організацію даних, пов'язаних із програмними проєктами, а також взаємодію з іншими компонентами системи.Одним із ключових елементів є клас, відповідальний за аналіз програмного коду. Саме він реалізує методи обчислення метрик якості, формування результатів аналізу та створення узагальнених показників для проєкту. Виділення аналітичної логіки в окремий компонент дозволяє зробити систему більш гнучкою та спрощує подальше додавання нових критеріїв

оцінювання. Крім аналізу даних, у системі передбачено формування звітів про результати оцінювання і з цією метою я реалізував окремий сервісний клас, який відповідатиме за узагальнення отриманих результатів та за їх підготовку до відображення у веб-інтерфейсі.

Для глибшого розуміння послідовності виконання основних операцій у системі я побудував діаграму діяльності (див. рис. 2.5), яка і відображатиме процес аналізу якості програмного кода.

Рисунок 2.5 – Діаграма діяльності процесу аналізу якості програмного коду

На рисунку 2.5 я показав діаграму діяльності, за допомогою якої продемонструю послідовність виконання основних операцій під час аналізу якості

програмного коду. Отже, процес починається із завантаження даних про зміни у коді. Після цього система виконує перевірку коректності отриманої інформації. У випадку успішної перевірки здійснюється обчислення метрик якості, формування результатів аналізу та їх збереження у базі даних. На основі отриманих результатів програмою формується узагальнений звіт, який для користувача відображається через вебінтерфейс. Якщо ж під час перевірки буде виявлено помилки або некоректні дані, то програма сформує повідомлення відповідного змісту. Таким чином, представлена діаграма дозволить наочно відобразити основну бізнес-логіку розроблюваного програмного рішення та продемонструє взаємозв’язок між окремими етапами аналізу.

Далі, щоб продемонструвати реалізацію основної логіки програмної системи, нижче я надам приклади ключових класів та методів, які дадуть змогу забезпечити аналіз якості програмного коду та формування результатів оцінювання.

Для реалізації логіки аналізу якості програмного коду я створив сервісний клас AnalysisService. Він безпосередньо відповідатиме за обчислення основних метрик та формування результатів аналізу.

Лістинг 2.1 – Сервісний клас AnalysisService, який відповідає за обчислення основних метрик та формування результатів аналізу

```
class AnalysisService {
    calculateMetrics(commitData) {
        const metrics = {
            complexity: this.calculateComplexity(commitData),
            duplication: this.calculateDuplication(commitData),
            maintainability: this.calculateMaintainability(commitData)
        };

        return metrics;
    }

    calculateComplexity(commitData) {
        return commitData.linesOfCode * 0.1;
    }

    calculateDuplication(commitData) {
        return commitData.duplicateBlocks * 2;
    }

    calculateMaintainability(commitData) {
        return 100 - (commitData.errors * 5);
    }
}
```

Наведений на лістингу 2.1 клас демонструє підхід до реалізації аналітичної частини системи. Метод calculateMetrics виконує узагальнення окремих показників якості. Допоміжні методи відповідають за обчислення конкретних метрик. В свою чергу виділення логіки аналізу в окремий сервісний компонент спрощує підтримку системи та дозволяє розширювати набір критеріїв оцінювання.

Клас Project реалізовано для роботи з інформацією про програмні проекти. Він забезпечує збереження основних даних та взаємодію зі змінами у коді.

Лістинг 2.2 – Клас Project для роботи з інформацією про програмні проєкти.

```
class Project {
    constructor(name, description) {
        this.name = name;
        this.description = description;
        this.commits = [];
    }

    addCommit(commit) {
        this.commits.push(commit);
    }

    getCommitCount() {
        return this.commits.length;
    }
}
```

Клас Project використовується для представлення програмного проєкту в системі. Він забезпечує зберігання інформації про проєкт, а також роботу зі списком комітів, що пов'язані з відповідними змінами у коді. Такий підхід забезпечить централізацію управління даними про проєкт.

Клас User я реалізував для представлення учасників системи та їх ролей .

Лістинг 2.3 – Клас User – представлення учасників системи та їх ролей.

```
class User {
    constructor(name, role) {
        this.name = name;
        this.role = role;
    }

    getUserInfo() {
        return `${this.name} (${this.role})`;
    }
}
```

Представлений в лістингу 2.3 клас забезпечує роботу з інформацією про користувачів системи. Він дозволяє зберігати дані про роль користувача та використовувати їх під час взаємодії з іншими компонентами вебдодатка.

Клас ReportService реалізовано з метою узагальнення результатів аналізу та їх підготовки до відображення.

Лістинг 2.4 – Клас ReportService для формування узагальнених результатів аналізу.

```javascript
class ReportService {
    generateProjectReport(results) {
        return {
            averageScore: this.calculateAverage(results),
            totalCommits: results.length
        };
    }

    calculateAverage(results) {
        const total = results.reduce((sum, item) => sum + item.value, 0);
        return total / results.length;
    }
}
```

Клас ReportService використовується саме для формування узагальнених результатів аналізу. Також він створює звіти стосовно стану програмного проєкту. Виділивши цю функціональність в окремий компонент я зможу відокремити логіку обробки даних від механізмів їх представлення.

Таким чином, наведені приклади продемонструють загальний підхід до реалізації програмної системи. Також я показав організацію основної логіки. Використання окремих сервісних компонентів для аналізу даних та формування звітів дозволять зробити систему більш структурованою. Крім того, це дасть змогу спростити подальший розвиток моєї розробки.В свою чергу розділення функціональності між класами забезпечить кращу підтримку програмного коду, зменшить залежність між компонентами та створить можливість розширення вебзастосунку, але без суттєвого перепроєктування вже реалізованої його логіки.

Підсумовуючи ,в підрозділі 2.6 я розглянув реалізацію основних класів та методів моєї розробки. Також представив процес виконання аналізу якості програмного коду. Важливо що саме використання парадигм об’єктно-орієнтованого підходу дозволило мені розділити функціональність системи на окремі логічні компоненти. Також я зміг забезпечити гнучку організацію програмного коду. В цілому ж реалізовані класи та методи формують основу моєї програмної системи та забезпечують виконання аналізу якості програмного коду у межах командних програмних проєктів.

2.7 Розробка інтерфейсу користувача

Далі, вже після виконання реалізації серверної логіки розроблюваного програмного забезпечення я перейшов до наступного етапу. Розробка інтерфейса користувача, забезпечить взаємодію користувача з функціоналом системи аналізу якості програмного коду. Основна мета цього етапу – це створити просте, логічно та інтуїтивно зрозуміле середовище. В ньому користувач зможе працювати з та над проєктами, переглядати результати аналізу та отримувати узагальнені показники якості коду. Під час проєктування самого інтерфейса основний акцент я робив на зручності навігації та мінімізації зайвих елементів. Так як моя розробка все таки має більш аналітичний характер, для мене важливим було забезпечити чітку структуру подачі інформації. Необхідно щоб користувач міг швидко орієнтуватися у даних та не витрачати час на пошук потрібних функцій.

Таким чином побудову інтерфейсу я реалізовував за принципом розділення на функціональні зони. Зокрема, у верхній або бічній частинах розташував навігаційне меню, яке забезпечує доступ до основних розділів системи, а саме: перегляду проєктів, результатів аналізу, звітів та налаштувань. Обраний підхід дозволить мені забезпечити швидкий доступ до ключових функцій без перевантаження робочої області. Центральна частина інтерфейсу використовується для відображення основного контенту. Тут користувач отримає інформацію про обраний проєкт, результати аналізу програмного коду та

узагальнені метрики якості. Усі дані будуть подаватися у структурованому вигляді. Це дозволить швидко оцінити стан програмного проекту та зробити висновки щодо якості коду.Окремо я передбачив блок для зведених результатів. У ньому відображатимуться узагальнена оцінка та рекомендації. Це дозволить користувачеві не лише бачити окремі метрики, а й отримувати загальне уявлення про стан програмного продукту.

Для того, щоб продемонструвати основні функціональні області та їх взаємозв'язок, для наочного відображення структури взаємодії користувача із системою я побудував діаграму інтерфейсу вебдодатка. На рисунку 2.6 (див. додаток) представлено структуру інтерфейсу користувача системи аналізу якості програмного коду.

На рисунку 2.6 я показав загальну структуру інтерфейсу вебзастосунка. Інтерфейс я умовно поділив на дві основні частини:

1-ша навігаційна область (істить основні розділи системи, через які користувач може переходити між проєктами, результатами аналізу, звітами та налаштуваннями. Це дозволить швидко отримувати доступ до необхідного функціоналу без ускладнення структури інтерфейсу);

2-га робоча область користувача(використовується для відображення інформації про програмний проєкт, результатів аналізу та узагальнених показників якості коду. Інформація подається у вигляді окремих логічних блоків, що спрощує її сприйняття та дозволяє користувачу зосередитися на ключових результатах оцінювання).

Обраний підхід до організації інтерфейсу забезпечить зручність роботи із системою, спростить навігацію між функціональними модулями та створить можливість подальшого розширення функціональності веб-додатку без суттєвої зміни структури інтерфейсу.

Для демонстрації реалізації окремих елементів інтерфейсу нижче (див.лістинг 2.5) я наведу приклад компонента. Він використовуватиметься для відображення результатів аналізу програмного коду.

Лістинг 2.5 – Клас ReportService для формування узагальнених результатів аналізу.

```
function AnalysisResults({ metrics }) {
    return (
        <div className="results-block">
            <h2>Результати аналізу</h2>

            <p>Складність коду: {metrics.complexity}</p>

            <p>Дублювання коду: {metrics.duplication}</p>

            <p>Підтримуваність: {metrics.maintainability}</p>
        </div>
    );
}
```

Наведений компонент використовуватиметься для відображення результатів аналізу якості програмного коду у вебінтерфейсі. Дані він отримуватиме у вигляді набору метрик та формуватиме структуроване представлення результатів вже для користувача.

Використання компонентного підходу є оптимальним для моєї реалізації і виконання програмою постевлених задач, оскільки дозволить мені повторно застосовувати окремі елементи інтерфейсу в різних частинах системи. Також це спростить підтримку та модифікацію клієнтської частини вебзастосунка.

Підсумовуючи, в підрозділі 2.7 моєї роботи я розглянув процес розробки інтерфейсу користувача вебзастосунку основним призначенням якого є аналіз якості програмного коду. Я сформував структуру основних сторінок системи, реалізував компонентний підхід до побудови інтерфейсу. Також я забезпечив зручне представлення результатів аналізу. таким чином, розроблений інтерфейс дозволить забезпечити ефективну взаємодію користувача із системою та буде підтримувати виконання основних функцій програмного забезпечення.

3 ТЕСТУВАННЯ, ВПРОВАДЖЕННЯ ТА ПІДТРИМКА

Після завершення етапів проєктування та реалізації розроблюваного програмного забезпечення, важливі моменти яких я описав в розділі 2, наступним важливим кроком стала перевірка його працездатності в умовах практичного використання. На цьому етапі особливу увагу я приділив тестуванню функціональних можливостей та перевірці стабільності роботи окремих компонент. Також важливим для мене було зосереити увагу на оцінці коректності отриманих результатів аналізу.

Окрім безпосереднього тестування, мені було важливо дослідити особливості розгортання програмного забезпечення, визначення системних вимог та перевірка відповідності реалізованої системи поставленим завданням. Це дозволить мені оцінити готовність рзробки вже до безпосереднього використання у межах та з врахуванням специфіки командних програмних проєктів, а також підтвердити ефективність реалізованих рішень [40-42].

3.1 Тестування вебзастосунку

Після завершення основних етапів розробки заплановане проведення тестування реалізованих модулів. Основною метою тестування буде перевірка коректності роботи реалізованого функціоналу, стабільності взаємодії між компонентами протопиту системи та відповідності отриманих результатів поставленим вимогам. Для системи аналізу якості програмного коду етап тестування я вважаю особливо важливим, оскільки навіть незначні помилки у роботі аналічних модулів можуть повпливати на достовірність результатів оцінювання та формування звітів.

Під час тестування змодельованих сценаріїв основну увагу я приділяв не лише окремим функціям вебзастосунку, а й його взаємодії між клієнтською та серверною частинами системи. Я також перевіряв коректність обробки даних, передачу інформації між модулями, формування результатів аналізу та їх

відображення у користувацькому інтерфейсі. Окремо оцінював стабільність роботи прототипу системи при виконанні типових сценаріїв використання. При тестуванні я також перевіряв роботу функцій, пов’язаних із додаванням програмних проєктів, обробкою даних про зміни у коді, розрахунком метрик якості, формуванням узагальнених результатів та відображенням інформації у веб-інтерфейсі. Також я тестував коректність працездатності системи при введенні некоректних або неповних даних. Це робив тому, що стабільність обробки помилок є важливою складовою надійності програмного забезпечення.Окрему увагу я приділив перевірці інтерфейсу користувача. Під час цього тестування оцінював зручність навігації, коректність відображення елементів інтерфейсу та зрозумілість подання результатів аналізу. Це дозволило мені переконатись, що система може використовуватися не лише як технічний інструмент для аналізу програмного коду, а і як зручне середовище для роботи учасників командних програмних проєктів.

В цілому ж процес тестування проводив поступово, починаючи з перевірки окремих компонентів системи та завершуючи комплексною перевіркою роботи в цілому моєї розробки. Такий підхід дав мені иожливість виявити помилки ще на ранніх етапах та уникнути накопичення проблем у процесі інтеграції окремих модулів.

Таким чином в результаті проведеного тестування я зможу підтвердити працездатність основних функцій програмної системи, стабільність взаємодії між її компонентами та коректність формування результатів аналізу якості програмного коду. Отримані результати дадуть мені можливість перейти до наступних етапів впровадження та перевірки програмного забезпечення в умовах, наближених до реального використання.

3.1.1 Види та план тестування

Для перевірки працездатності прототипу програмної системи для аналізу якості програмного коду я провів комплексне тестування окремих модулів та основних функціональних її можливостей. Оскільки розробка складається з кількох взаємопов’язаних компонентів, тестування я проводив поетапно. Тобто, починав я з перевірки окремих функцій і до оцінки взаємодії між клієнтською та серверною частинами.

Під час тестування основну увагу я приділяв не лише правильності виконання окремих операцій, а й стабільності роботи застосунка вцілому. З цією метою я перевіряв процеси обробки даних, формування результатів аналіза, взаємодію між модулями та коректність відображення інформації у вебінтерфейсі. Також я, але вже окремо, оцінював реакція програмного рішення на некоректні дії користувача, оскільки саме опрацювання помилок є важливою складовою надійності не лише мсого, але й будь-якого якісного програмного забезпечення.

В цілому ж тестування я проводив на основі змодельованих сценаріїв використання, які відображають можливі, але типові дії користувача під час роботи із системою. Такий підхід дозволив мені оцінити роботу реалізованого прототипу в умовах, наближених до практичного використання, а також виявити потенційні недоліки у взаємодії між окремими компонентами вбзастосунку.

Для тестування я використав декілька його видів. При цьому кожен із них виконував окреме завдання та дозволяв оцінити різні аспекти роботи програмної системи. Далі, в таблиці 3.1, я наведу основні види застосованого для моєї програмної системи тестування.

Таблиця 3.1 – Основні види тестування програмної системи

№ п/п	Вид тестування	Призначення тестування	Що перевірялося при тестуванні
1	Функціональне тестування	Перевірка реалізації основних функцій системи	Робота модулів аналізу, формування звітів, обробка даних
2	Інтеграційне тестування	Перевірка взаємодії між компонентами системи	Передача даних між серверною та клієнтською частинами
3	Тестування інтерфейсу	Оцінка зручності та коректності відображення данних	Навігація, відображення результатів, структура сторінок
4	Тестування обробки помилок	Перевірка реакції системи на некоректні дії	Введення порожніх або неправильних даних
5	Навантажувальне тестування	Оцінка стабільності роботи при збільшенні кількості даних	Обробка результатів аналізу та формування звітів
6	Модульне тестування	Перевірка окремих функціональних компонентів	Робота методів розрахунку метрик та аналізу коду

В таблиці 3.1 я навів види тестування, які використав комплексно, так як перевірка лише окремих функцій не дозволить мені повністю оцінити працездатність системи. Для прикладу, коректна робота модулю для аналізу не гарантує правильного відображення результатів у вебінтерфейсі, тому окремо проводилася перевірка взаємодії між компонентами системи.

Особливу увагу я приділив функціональному тестуванню, оскільки саме воно дозволяло перевірити реалізацію ключових можливостей вебдодатку. У межах такого тестування я оцінював коректність обробки інформації про програмні проєкти, формування метрик якості та побудову узагальнених результатів аналізу. В свою чергу інтеграційне тестування я використовував для перевірки передачі даних між серверною частиною та клієнтським інтерфейсом. Під час перевірки я оцінював правильність обміну даними між модулями системи, а також стабільність роботи API при виконанні основних сценаріїв використання.

Окремо я проводив тестування користувацького інтерфейсу. Перевіряв логіка навігації між сторінками, коректність відображення результатів аналізу та зручність взаємодії користувача із системою. Це дозволило мені оцінити не лише технічну складову вебдодатку, а і його практичну зручність для потенційних користувачів. Також я провів перевірку роботи системи при введенні помилкових або неповних даних. Такий підхід дозволив оцінити стійкість програмного забезпечення до некоректних дій користувача та перевірити механізми обробки помилок.

Для систематизації процесу тестування я сформував план перевірки основних функціональних модулів системи. Послідовність виконання тестування наведу далі в таблиці 3.2.

Таблиця 3.2 – План проведення тестування

№ п/п	**Етап тестування**	**Основна мета тестування**
1	Перевірка окремих модулів	Виявлення помилок у роботі окремих функцій
2	Перевірка взаємодії компонентів	Аналіз передачі даних між модулями
3	Тестування користувацького інтерфейсу	Перевірка навігації та відображення інформації
4	Перевірка обробки помилок	Аналіз реакції системи на некоректні дії
5	Комплексне тестування системи	Перевірка стабільності роботи веб-додатку

Таким чином, застосування поетапного підходу до тестування дозволило мені поступово перевірити як окремі компоненти системи, так і взаємодію між ними. Це дало можливість своєчасно виявити недоліки реалізації, оцінити стабільність роботи програмного забезпечення та підтвердити працездатність реалізованого прототипу системи аналізу якості програмного коду.

3.1.2 Розробка тестових сценаріїв

Після визначення основних видів тестування та формування загального плану перевірки програмної системи (підрозділ 3.1.1) я розробив тестові сценарії, які використав для оцінки працездатності реалізованих функціональних модулів мого вебдодатку. Основною метою створення тестових сценаріїв стала перевірка поведінки системи під час виконання типових дій користувача. Також важливим є аналіз реакції програмного забезпечення на різні варіанти введених данних.

В загальному вартарта зазначити ,що тестові сценарії я формував на основі варіантів використання, які визначені у першому розділі роботи. Такий підхід дозволив мені забезпечити зв’язок між функціональними вимогами до системи та процесом перевірки її на працездатність. Під час розробки сценаріїв основну увагу я приділяв не лише успішному виконанню окремих операцій, а й акцентувався на можливі помилкові ситуації, які можуть, і ймовірно будуть, виникати під час роботи користувача із системою.

Під час тестування я перевіряв і самі процеси створення програмного проекту, обробки інформації про зміни у коді, розрахунку метрик якості, формування результатів аналізу та відображення даних у користувацькому інтерфейсі. Окремо я оцінював реакцію системи на некоректне введення даних або порушення послідовності виконання дій. А вже для систематизації процесу перевірки сформував набір тестових сценаріїв (див. таблицю 3.3), які охоплюють основні функціональні можливості реалізованого прототипу системи.

Таблиця 3.3 – Основні тестові сценарії

№ п/п	**Назва сценарію**	**Дія користувача**	**Очікуваний результат**
1	Створення проєкту	Користувач додає новий програмний проєкт	Проєкт успішно створюється та відображається у системі

Продовження таблиці 3.3

№ п/п	Назва сценарію	Дія користувача	Очікуваний результат
2	Завантаження даних успішно	Користувач передає дані для аналізу	Дані обробляються системою
3	Аналіз якості коду	Запуск процесу аналізу	Формуються метрики якості програмного коду
4	Перегляд результатів	Відкриття сторінки результатів аналізу	Користувач отримує структуровані результати оцінювання
5	Формування звіту	Генерація узагальненого звіту	Система формує звіт із результатами аналізу
6	Введення некоректних даних	Передача порожніх або неправильних значень	Система повідомляє про помилку
7	Перехід між сторінками	Використання навігаційного меню	Сторінки відкриваються без помилок
8	Повторний аналіз проєкту	Повторний запуск аналізу	Результати оновлюються коректно

Наведені в таблиці 3.3 тестові сценарії дозволили мені перевірити основні функціональні можливості моєї розробленої програмної системи. Також вони дами мені можливість оцінити стабільність її роботи під час виконання типових користувацьких дій. Особливу увагу я приділив сценаріям, які пов’язані з формуванням результатів аналізу, адже саме вони є ключовою частиною системи аналізу якості програмного коду. Крім цього під час перевірки сценаріїв я оцінював не лише правильність отриманих результатів, а й "поведінку" інтерфейсу користувача, швидкість відображення інформації. Також важливо було оцінити і коректність взаємодії між окремими компонентами системи. Це дозволило мені здійснити комплексну оцінку працездатність реалізованого прототипу вебдодатку.

Окремо я перевіряв реакцію системи на помилкові дії користувача. Зокрема, аналізував обробку порожніх полів, некоректних значень та повторного виконання окремих операцій. Проведення таких перевірок дала мені можливість оцінити стійкість системи до помилок та в цілому забезпечити більш стабільну роботу програмного забезпечення.

Таким чином, в результаті виконання тестових сценаріїв я підтвердив коректність роботи основних функціональних модулів системи, стабільність взаємодії між її компонентами та правильність формування результатів аналізу якості програмного коду.

3.2 Розгортання реалізації та системні вимоги

Після завершення реалізації основних функціональних можливостей програми та проведення тестування наступним наступним короком я досліджував особливості її розгортання та визначення технічних вимог для стабільної роботи вебзастосунка. На цьому етапі для мене важливим було не лише перевірити можливість запуску реалізованого прототипу, а й оцінити, наскільки система може бути адаптовною до подальшого використання у межах навчального чи командного середовища.

Так як програмна система реалізована у вигляді вебзастосунка, процес її розгортання є значно простішим, якщо порівнювати з локальними програмними рішеннями. Тут користувачу вже немає необхідності встановлювати окремі програмні модулі чи виконувати складне налаштування клієнтської частини системи. Взаємодія із функціоналом додатку здійснюється через веббраузер. Це, в свою чергу, дозволить спростити доступ до системи та забезпечити більш зручне використання в межах командної роботи.

Під час розгортання основна увага приділялася коректній взаємодії між серверною частиною, базою даних та користувацьким інтерфейсом. Для роботи веб-додатку необхідно забезпечити запуск серверного середовища Node.js, налаштувати підключення до бази даних PostgreSQL та перевірити стабільність обробки HTTP-запитів між клієнтською і серверною частинами системи. Далі я опишу бульш детально перелічені взаємодії. Отже:

- серверна частина відповідає за обробку запитів, виконання логіки аналізу програмного коду, роботу з базою даних та формування результатів аналізу. Саме сервер забезпечує взаємодію між окремими модулями системи та виконує основні

операції обробки даних. Для запуску серверної частини використовується середовище Node.js, яке дозволить ефективно працювати із вебзапитами та забезпечує достатню гнучкість для подальшого розширення функціональності програмної розробки;

- база даних PostgreSQL використовується для збереження інформації про програмні проєкти, результати аналізу, метрики якості та інші службові дані. Використання реляційної системи керування базами даних дозволить менні забезпечити структуроване збереження інформації та спростить подальшу роботу з даними;

- клієнтська частина реалізована у вигляді вебінтерфейса, через який користувач отримує доступ до функцій системи. Для роботи з застосунком достатньо сучасного браузера. Це суттєво спрощує процес використання програмного забезпечення. Такий підхід також дозволить уникнути залежності від конкретної операційної системи та забезпечить можливість використання вебзастосунку на різних пристроях.

Під час перевірки розгортання я також окремо оцінював і стабільність взаємодії між компонентами системи. Перевіряв підключення до бази даних, коректність передачі даних через API, обробку запитів та відображення результатів аналізу у вебінтерфейсі. Завдяки такій перевірці я переконався та впевнився в тому, що реалізований прототип може стабільно працювати у локальному середовищі розробки та підтримує виконання передбачених основних функціональних можливостей.

Також я визначив та навів в таблиці 4.1 мінімальні системні вимоги, необхідні для запуску програмної системи.

Таблиця 3.4 – Мінімальні системні вимоги

Компонент	**Мінімальні вимоги**
Операційна система	Windows 10 / Linux / macOS
Процесор	двоядерний процесор від 2.0 GHz

Продовження таблиці 3.4

Компонент	**Мінімальні вимоги**
Оперативна пам’ять	від 4 GB RAM
Вільне місце на диску	від 2 GB
Середовище виконання	Node.js
Система керування базою даних	PostgreSQL
Веб-браузер	Google Chrome, Mozilla Firefox, Microsoft Edge
Підключення до мережі	необхідне для роботи веб-додатку

Наведені системні вимоги є оптимальними та дозволяють користувачеві використовувати систему навіть на звичайних персональних комп’ютерах, без використання спеціалізованого обладнання. Я вважаю, що це є важливою перевагою саме веборієнтованого підходу. Такий висновок я зробив, оскільки користувачеві не потрібно мати потужну технічну конфігурацію для роботи із системою аналізу якості програмного коду.

Окремо варто зазначити, що реалізований прототип може бути адаптований до подальшого розгортання на віддаленому сервері або у хмарному середовищі. Використання сучасних вебтехнологій та клієнтсерверної архітектури дозволяє масштабувати систему та забезпечити підтримку більшої кількості користувачів і програмних проєктів.

Таким чином, у межах даного етапу моєї кваліфікаційної роботи я визначив основні особливості розгортання програмної системи, перевірив працездатність реалізованого прототипу у локальному середовищі та сформував мінімальні системні вимоги, які є необхідними для стабільної роботи вебзастосунка основним призначенням якого є аналіз якості програмного коду.

3.3 Верифікація вебзастосунку

Після завершення тестування та перевірки основних функціональних можливостей вебзастосунку далі я провів верифікацію реалізованого локального програмного забезпечення. За основну мету цього етапу я брав перевірку відповідності розробленого вебзастосунку поставленим вимогам, які я визначив на початкових етапах проєктування (див. розділ 1 Аналіз вимог до вебзастосунку). Якщо тестування в першу чергу спрямовано на пошук помилок та оцінку стабільності роботи окремих компонентів, то вже верифікація дозволить мені оцінити, наскільки отриманий результат відповідатиме поставленому завданню та чи забезпечить систему виконання необхідних функцій.

Отже, під час проведення верифікації основну увагу я приділив перевірці реалізованих функціональних можливостей вебзастосунку, перевірці логіки взаємодії між окремими модулями та коректності відображення результатів аналізу програмного коду. Також я оцінював, чи забезпечить система можливість роботи із програмними проєктами, формування результатів аналізу, перегляд метрик якості та генерацію узагальнених звітів. Окремо я ще перевіряв відповідність інтерфейсу користувача початковим вимогам щодо зручності використання та структури навігації. Оскільки для систем аналітичного типу важливим є не лише отримання результатів, а і спосіб їх подання користувачу. Саме тому під час верифікації я оцінював зрозумілість структури вебзастосунку, логічність розташування основних елементів інтерфейсу та зручність переходу між функціональними модулями системи. Ще я оцінював відповідність реалізованої архітектури вимогам до масштабованості та можливості подальшого розширення функціональності. А саме використання клієнт-серверної архітектури, модульного підходу до побудови системи та окремого шару роботи з базою даних дозволять мені у майбутньому ще й доповнювати вебзастосунок новими функціями. При цьому не буде ніякої необхідності в суттєвих змінах вже реалізованих компонентів.

В цілому ж в процесі верифікації я зміг підтвердити, що реалізований прототип вебзастосунку забезпечить виконання основних функцій, визначених у межах поставленого завдання. Система підтримує роботу з програмними проєктами, формування результатів аналізу якості програмного коду, перегляд метрик та відображення узагальненої інформації у користувацькому інтерфейсі. А вже для узагальнення результатів верифікації далі (див. таблицю 3.5) я сформував таблицю відповідності основних вимог реалізованому функціоналу вебзастосунку.

Таблиця 3.5 – Відповідність функціональних вимог реалізованому вебзастосунку

Функціональна вимога	**Результат перевірки**
Створення та перегляд програмних проєктів	-реалізовано
Аналіз якості програмного коду	-реалізовано
Формування метрик якості	-реалізовано
Відображення результатів у вебінтерфейсі	-реалізовано
Формування узагальнених звітів	-реалізовано
Навігація між основними модулями	-реалізовано
Обробка помилкових дій користувача	-частково реалізовано
Масштабування функціональності системи	-передбачено архітектурою

Загалом, проаналізувавши можу констатувати, що наведені результати підтверджують, що реалізований вебзастосунок вцілому відповідає визначеним функціональним вимогам. Забезпечує виконання основних завдань аналізу якості програмного коду. Водночас окремі аспекти, пов’язані з розширеною обробкою помилок та оптимізацією окремих функцій, можуть бути доопрацьовані у процесі подальшого розвитку програмного продукту. Таким чином, проведена верифікація дозволила мені підтвердити працездатність реалізованого прототипу вебзастосунку і я зміг оцінити відповідність отриманого результату поставленим вимогам та визначити можливості подальшого вдосконалення системи.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

Безпека життєдіяльності та охорона праці є невід'ємними складовими професійної діяльності фахівця спеціальності 121 «Інженерія програмного забезпечення». Розробка, тестування та супровід програмних продуктів здійснюються переважно з використанням комп'ютерної техніки, що потребує дотримання вимог безпеки праці, ергономіки та санітарно-гігієнічних норм. Забезпечення належних умов праці сприяє збереженню здоров'я працівників, підвищенню ефективності роботи та зниженню ризиків виникнення професійних захворювань. У цьому розділі розглянуто основні аспекти організації безпечного робочого місця інженера-програміста та заходи щодо запобігання впливу шкідливих і небезпечних виробничих факторів.

4.1 Долікарська допомога при ураженні електричним струмом

Використання електричної енергії є невід'ємною складовою наукових досліджень, зокрема при роботі з комп'ютерною технікою. Незважаючи на відносно низькі рівні напруги в побутових ПК (220 В), неправильна експлуатація або технічні несправності можуть призвести до ураження електричним струмом. Дія електричного струму на організм людини залежить від типу струму, напруги, тривалості його проходження, шляху проходження, індивідуальних особливостей і оточуючого середовища [43].

Електричний струм, проходячи через тіло людини, викликає низку фізіологічних ефектів: судоми, зупинку серця, термічні опіки, електроліз тканин. У контексті експлуатації ПК виникають наступні ситуації:

- пошкодження ізоляції проводів живлення або блоку живлення;
- дотик до неізольованих елементів при розібраному корпусі ПК;
- використання несправних подовжувачів або розеток;
- висока вологість у приміщенні або відсутність заземлення.

Особлива увага приділяється дотриманню наступних правил при користуванні ПК чи ПЕОМ, відповідно до яких необхідно:

- використовувати справні мережеві фільтри з функцією захисту від перенапруги;
- уникати відкритого контакту з внутрішніми компонентами ПК, особливо при ввімкненому живленні;
- регулярно перевіряти заземлення комп'ютерної техніки;
- дотримуватись інструкцій виробника при використанні лабораторного устаткування;
- не працювати на вологих поверхнях або з мокрими руками.

Важливо пам'ятати, що навіть при низьких напругах можливе ураження струмом– особливо при підвищеній вологості або пошкодженій ізоляції [44].

Згідно з Порядком надання домедичної допомоги постраждалим при ураженні електричним струмом або блискавкою Наказу Міністерства охорони здоров'я України від 09 березня 2022 року №441, першочергові кроки до та при наданні домедичної допомоги включають [45]:

1) перед наданням допомоги переконатися у відсутності небезпеки для себе, оточуючих, постраждалого та тільки за її відсутності перейти до наступного кроку;

2) якщо постраждалий у свідомості, заспокоїти та пояснити свої наступні дії;

3) здійснити виклик екстреної медичної допомоги та дотримуватись вказівок диспетчера прийому виклику;

4) при ураженні постраждалого електричним струмом:

- якщо постраждалий без свідомості, впевнитись, що дія електричного струму на постраждалого припинена;
- всі дії по припиненню дії електричного струму слід здійснювати за умови проходження відповідного навчання або здійснити виклик за єдиним телефонним номером системи екстреної допомоги населенню 112;

• якщо дія електричного струму на постраждалого припинена, слід надати йому домедичну допомогу, відповідно до наявних пошкоджень;

5) забезпечити постійний нагляд за постраждалим до приїзду бригади екстреної (швидкої) медичної допомоги;

6) при погіршенні стану постраждалого до приїзду бригади екстреної (швидкої)медичної допомоги повторно здійснити виклик екстреної медичної допомоги;

7) за можливості зібрати у постраждалого чи оточуючих максимально можливу інформацію стосовно обставин отримання травми. Всю отриману інформацію передати працівникам бригади екстреної (швидкої) медичної допомоги або диспетчеру служби екстреної медичної допомоги [45].

Знання та практичні навички надання долікарської допомоги при ураженні електричним струмом є необхідними для кожного, хто працює з електронним обладнанням, особливо в рамках розробки наукових програмних продуктів. Врахування електробезпеки під час роботи з ПК не лише зберігає здоров'я, а й запобігає втратам обладнання та зупинці дослідницького процесу.

4.2 Інженерно-технічні рішення з охорони праці

Розробка та експлуатація програмного комплексу для математичного моделювання енергій активації у цеолітах передбачає інтенсивне використання комп'ютерної техніки. Важливо забезпечити безпечні електро-технічні, ергономічні та інженерно-технічні умови не лише для збереження здоров'я персоналу, а й неперервності дослідницького процесу.

У межах охорони праці під час проєктування та експлуатації програмного комплексу впроваджуються такі інженерно-технічні рішення, відповідно до ДСТУEN ISO 11064-5:2017 та ДСТУ EN ISO 9241-13:2017 [46, 47]:

1) оснащення робочого місця , а саме використання :

• використання ергономічного крісла з відповідною висотою, шириною, глибиною та кутом сидіння, щоб забезпечувати зміни позий та достатній комфорт для ефективного виконання завдання;

• використання стола з регулюванням висоти, який має дозволяти зручне та ефективне положення верхньої частини рук, передпліч і кистейрук;

• регулювання монітора на рівні очей, з дотриманням кута зору(оптимальний0°) не більше 40° будь-де на активній області дисплею:

2) організація проводки, а саме:

• з'єднання надійно закріплені так, щоб вони не спричиняли небезпеки, тягнучись упоперек робочої поверхні чи підлоги;

• довжина кабелів достатня, щоб задовольнити фактичні і прогнозовані потреби користувачів;

• проводка здатна забезпечити повний діапазон регулювання, якщо є регульовані поверхні;

3) утримання провітрюваного приміщення з рівномірним освітленням: ефективний рух повітря (щоденне провітрювання або припливно-витяжна вентиляція) та рівномірне, без тіней, освітлення —важлива частина забезпечення нормального мікроклімату та профілактики перевтоми очей;

4) регламентація режиму праці й відпочинку шляхом впровадження перерви для відпочинку тривалістю 15 хвилин через кожні дві години, впровадження вправ для очей та м’язів верхніх кінцівок [46, 47];

5) дотримання оптимально мікроклімату та чистоти: оптимально18–24°С,вологість – ≤ 60 %; очистка підставок, корпусів та вентиляційних каналів для забезпечення електротехнічної безпеки і стабільності роботи електроніки;

6) використання звукопоглинальних матеріалів для обробки стін, стель і підлоги в приміщеннях з декількома комп’ютерними станціями для зниження шуму [48].

Інженерно-технічні заходи передбачають не лише створення безпечного середовища, а й:

- проведення інструктажу з охорони праці;
- навчання основам електробезпеки, правил надання домедичної допомоги при ураженні струмом [49].

Варто зазначити, що ПК загального призначення не використовуються в умовах підвищеної небезпеки, тому експлуатація комп'ютерної техніки такого типу не вважається роботою з підвищеною небезпекою. Отже, впроваджені інженерно-технічні рішення не враховують заміну небезпечних матеріалів найменш небезпечні в проєктуванні. Також розробка окремої інструкції з охорони праці при використанні комп'ютерної техніки є недоцільною, достатньо використати чи розробити інструкцію з елекробезпеки, яка буде враховувати специфіку обладнання, що використовується. Використання сучасних інженерно-технічних рішень, згідно з нормами інтер'єру, мікроклімату та акустики, має не лише утримувати наукове обладнанняв працездатному стані, але й створювати безпечне середовище для персоналу

ВИСНОВКИ

Метою роботи на здобуття освітнього рівня «бакалавр» була розробка вебзастосунку для комплексного аналізу якості програмного коду із забезпеченням представлення результатів та врахуванням особливостей та специфіки саме командної взаємодії розробників. Тобто, ціллю було створити інструмент, який дозволив би "бачити" не лише окремі помилки в коді, а й оцінити загальний стан якості проєкту, створеного силами всієї залученої команди розробників.

Для досягнення поставленої мети я вирішив наступні задачі:

1. Проаналізував сучасні підходи та інструменти оцінювання якості програмного коду і визначив їхні обмеження в умовах командної розробки.

2. Дослідив основні показники якості програмного коду.

3. Визначив доцільні для застосування метрики.

4. Сформував функціональні та не функціональні вимоги врахувавши потреби саме командної роботи.

5. Розробив архітектуру програмного рішення, яка забезпечує інтеграцію різних методів аналізу коду та обробку результатів.

6. Реалізував вебзастосунок для аналізу якості програмного коду з можливістю збору, обробки та представлення даних.

7. Забезпечив підтримку командної взаємодії, зокрема відображення внеску (частки) роботи кожного розробника, змін у коді та динаміки якості.

8. Провів тестування розробленого застосунку.

9. Оцінив ефективність у задачах аналізу якості програмного коду створеного в результаті роботи групи розробників.

Після реалізації проєкту я отримав наступні результати:

-інтегрований (комплексний) підхід до оцінювання якості програмного коду, який передбачає об’єднання різнорідних метрик у єдину узгоджену систему з можливістю їх спільної інтерпретації;

-можливість подальшого розвитку підходу до аналізу якості коду за рахунок врахування специфіки роботи команди, зокрема розподілу внесення змін між розробниками, частоти модифікацій та повторюваності типових помилок;

- можливість відстуження в часі та розгляду якості програмного коду як динамічної характеристики;

- модель представлення результатів аналізу, яка дозволить поєднати технічні показники якості з інформацією про процес розробки;

- вдосконалення підходу до візуалізації результатів аналізу, орієнтований на швидке виявлення проблемних ділянок коду та закономірностей їх виникнення при командній роботі;

- концепцію інтегрального показника якості програмного коду, що дозволить формувати узагальнену оцінку стану програмного проєкту на основі сукупності технічних метрик та параметрів командної розробки.

Новизна одержаних результатів полягає у розробці підходу до аналізу якості програмного коду, який, на відміну від існуючих рішень, розглядає якість не як сукупність окремих перевірок, а як комплексний аналіз якості програмного коду (модель–динаміка–команда–інтегральний/комплексний індекс), що формується у процесі командної розробки.

Роботу апробовано, а результати було представлено під час роботи конференції 24-25 квітня 2026 року та опубліковано в матеріалах IX Міжнародної студентської науково-технічної конференції ТНТУ імені Івана Пулюя “Природничі та гуманітарні науки. Актуальні питання”.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Методичні вказівки до виконання кваліфікаційної роботи бакалавра для здобувачів спеціальності 121 – Інженерія програмного забезпечення, всіх форм навчання / укладачі: Михалик Д.М., Цуприк Г.Б., Бревус В.М. – Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2024. – 45 с. (https://elartu.tntu.edu.ua/handle/lib/50317)

2. ISO/IEC 25010:2023 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Product quality model. URL: ISO/IEC 25010:2023

3. ISO/IEC 25010:2011 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models. URL: ISO/IEC 25010:2011

4. ISO/IEC 9126-1:2001 Software engineering — Product quality — Part 1: Quality model. URL:ISO/IEC 9126-1:2001

5. Hamretskyi, R., & Gnatyuk, V. (2025). REVIEW AND ANALYSIS OF METHODS AND MODELS FOR ASSESSING THE QUALITY OF SOFTWARE IN INFORMATION AND COMMUNICATION SYSTEMS. *Science-Based Technologies*, *64*(4), 435–448. https://doi.org/10.18372/2310-5461.63.19752

6. Lee M.-C. Software quality factors and software quality metrics to enhance software quality assurance. British Journal of Applied Science & Technology. 2014. Vol. 4, No. 21. P. 3069–3095.

7. Гапон А. О. Методи та засоби статичного та динамічного аналізу коду / А. О. Гапон, В. М. Федорченко, О. В. Сєвєрінов // Радіотехніка : Всеукр. міжвід. наук.-техн. зб. – 2023. – Вип. 212. – С. 7–13. – DOI:10.30837/rt.2023.1.212.01.

8. SonarQube [Електронний ресурс]. Режим доступу: https://www.sonarsource.com/

9. ESLint [Електронний ресурс]. Режим доступу: https://eslint.org/

10. Pylint [Електронний ресурс]. Режим доступу: https://pylint.readthedocs.io/

11. Checkstyle [Електронний ресурс]. Режим доступу: https://checkstyle.sourceforge.io/

12. PMD [Електронний ресурс]. Режим доступу: https://pmd.github.io/

13. Code Climate [Електронний ресурс]. Режим доступу: https://codeclimate.com/

14. Snyk [Електронний ресурс]. Режим доступу: https://snyk.io/

15. DeepSource [Електронний ресурс]. Режим доступу: https://deepsource.com/

16. Mishra A., Linh N. T. D., Bhardwaj M., Pinto C. M. A. Multi-Criteria decision models in software reliability: Methods and Applications. Information Technology, Management and Operations Research Practices. 2022. DOI: 10.1201/9780367816414.

17. Романовський О. Г., Шаполова В. В., Квасник О. В., Гура Т. В. Психологія тімбілдингу : навчальний посібник / за заг. ред. Романовського О. Г., Калашникової С. В. Харків: «Друкарня Мадрид», 2017. 92 с. URL: http://repository.kpi.kharkov.ua/bitstream/KhPI-Press/36676/1/Book_2017_Romanovskyi_Psykholohiia_tymbildynhu.pdf.

18. Горбунова В. В. Психологія командотворення: ціннісно-рольовий підхід до формування та розвитку команд : монографія. Житомир: Вид-во ЖДУ ім. І.Франка, 2014. 380 с. URI: http://er.ucu.edu.ua/bitstream/handle/1/1044/Gorbunova_Psychology%20of%20teem%20building.pdf?sequence=1&isAllowed =y.

19. Алексенко О. В. Технології програмування та створення програмних продуктів : конспект лекцій. Суми: Сумський державний університет, 2013. 133 с. URL: https://essuir.sumdu.edu.ua/bitstream-download/123456789/30254/1/Alekseenko_Programuvannja.pdf;jsessionid=D575D30AC6AACA2D32533552C613A680.

20. Якими метриками IT-компанії міряють ефективність ШІ і де ще є проблеми [Електронний ресурс]. Режим доступу: https://dou.ua/lenta/articles/ai-metrics-it-companies/

21. Олянін, Д., Цуприк, Г. (2025) Transformer Neural Networks in Industry 4.0 / Д. Олянін, Г. Цуприк, Т. Говорущенко, О. Багрій-Заяць, І. Андрущак // Computer Information Technologies in Industry 4.0: proceedings of the 3rd International Workshop (CITI-2025), Ternopil, Ukraine, 11–12 June 2025. – Ternopil : Ternopil Ivan Puluj National Technical University, 2025 (Scopus) https://ceur-ws.org/Vol-4057/

22. Tsupryk, H., Olianin, D. (2025). Vydobuvannia danyh z tekstu vykorystovuiuchy transformerni neironni merezhi [Data extraction from text using Transformer Neural Networks]. Information Technology: Computer Science, Software Engineering and Cyber Security, 125–130, DOI: https://doi.org/10.32782/IT/2025-2-13

23. ОЛЯНІН D., & ЦУПРИК Н. (2025). Огляд ролі трансформерних нейроних мереж у видобувані інформації із неструктурованих даних. Measuring and computing devices in technological processes, 82(2), 360–364. https://doi.org/10.31891/2219-9365-2025-82-52

24. Tsupryk H. LLM-based Extraction from Resumes / D. Olianin, H. Tsupryk // Advanced Technologies in Scientific Research: collection of scientific papers with proceedings of the 1st International Scientific and Practical Conference, Rotterdam, Netherlands, 20–22 August 2025. – International Scientific Unity, 2025. – 72-76

25. Yaroslav Kotov, Evhenia Yavorska, Halyna Tsupryk, Róża Dzierżak 1 , Oleksandr Reshetnik, Viktoriia Bokovets (2025) Evaluating interoperability and data quality in FHIR-based AI assessment pipelines. Proc. SPIE 14009, Photonics Applications in Astronomy, Communications, Industry, and High Energy Physics Experiments 2025, 140091F (30 December 2025) https://doi.org/10.1117/12.3100561

26. Веб-розробка [Електронний ресурс]. Режим доступу: https://skillsbuild.org/uk/adult-learners/explore-learning/web-developer

27. Функціональні та нефункціональні вимоги (з прикладами) [Електронний ресурс]. Режим доступу: https://visuresolutions.com/uk/alm-guide/https://visuresolutions.com/uk/alm-guide/функціональні-та-нефункціональні-вимоги/

28. Литвин В.В., Пасічник В.В. , Шаховська Н.Б. Проектування інформаційних систем. Навчальний посібник. Львів: Магнолія 2006.- 2021 – 380 с.

29. Ізмайлова О.В. Проектування інформаційних систем: навч. посіб. Київ: КНУБА, 2023. – 88с.

30. Ушенко Ю.О., Ковальчук М.Л., Гавриляк М.С., Негрич А.Л. Методологія інформаційних систем та баз даних: теоретичний і практичний підходи: навч. посібник. Чернівці: ЧНУ ім. Ю. Федьковича.- 2021. – 240 с.

31. Топ-10 методологій розробки програмного забезпечення: переваги та недоілки [Електронний ресурс]. Режим доступу: https://whileweb.com/uk/blog/top-10-metodologij-rozrobki-programnogo-zabezpechennya-perevagi-ta-nedoliki/

32. Типи баз даних: особливості, відмінності та приклади [Електронний ресурс]. Режим доступу: https://dou.ua/lenta/articles/types-of-databases/

33. A. O. Kurotych, L. V. Bulatetska, Optimizing the process of ER diagram creation with PlantUML, CEUR Workshop Proceedings (2025) 47–57. URL: https://ceur-ws.org/Vol-3917/paper12.pdf

34. GitHub · Build and ship software on a single, collaborative platform · GitHub.
URL: https://raw.githubusercontent.com/kurotych/sqlant/b2e5db9ed8659f281208a687a344b34ff38129cd/puml-lib/db_ent.puml

35. PlantUML. PlantUML.com. URL: https://plantuml.com/

36. Welcome to The Hitchhiker’s Guide to PlantUML! – The Hitchhiker's Guide to PlantUML documentation. Crashedmind GitHub. URL: https://crashedmind.github.io/PlantUMLHitchhikersGuide/

37. Flanagan D. JavaScript: The Definitive Guide. — 7th ed. — Sebastopol : O'Reilly Media, 2020. — 704 p.

38. Tilkov S., Vinoski S. Node.js: Using JavaScript to Build High-Performance Network Programs // IEEE Internet Computing. – 2010. – Vol. 14, No. 6. – P. 80–83.

39. Node.js Documentation [Електронний ресурс]. – Режим доступу: https://nodejs.org/docs/latest/api/

40. Stapp L., Roman A., Pilaeten M. *ISTQB*® Certified Tester Foundation Level: A Self-Study Guide Syllabus v4.0. — Cham : Springer, 2024. — 406 p. [Електронний ресурс]. – Режим доступу: https://istqb.org/

41. Thompson G., Morgan P., Samaroo A. Software Testing: An ISTQB-BCS Certified Tester Foundation Level Guide (CTFL v4.0). — 5th ed. — Swindon : BCS Learning & Development Ltd, 2024. — 288 p.

42. Postman Learning Center [Електронний ресурс]. – Режим доступу: https://learning.postman.com/

43. Желібо Є. П. Безпека життєдіяльності. навч. посіб. / Є.П.Желібо, Н. М. Заверуха, В. В. Зацарний. 6-те вид. Київ : Каравела, 2023. 344с.

44. ТЕМА 17. Електробезпека. URL:https://dl.tntu.edu.ua/content.php?cid=289206

45. Про затвердження порядків надання домедичної допомогиособамприневідкладних станах: наказ МОЗ України від 09.03.2022№441. URL:https://zakon.rada.gov.ua/laws/show/z0356-22/conv#n800

46. ДСТУ EN ISO 11064-5:2017. Проектування центрівкеруванняергономічне. Частина 5. Засоби відображення інформації та органикерування(ENISO 11064-5:2008, IDT; ISO 11064-5:2008, IDT). Чинний від 2019-1-1. Вид.офіц.Київ : УкрНДНЦ, 2017.

47. ДСТУ EN ISO 9241-13:2017 Ергономічні вимогидороботизвідеотерміналами в офісі. Частина 13. Настанова щодо використання(ENISO9241-13:1998, IDT; ISO 9241-13:1998, IDT). Чинний від 2019-1-1. Вид. офіц. Київ:УкрНДНЦ, 2017.

48. Навчально-методичний посібник до практичних заняттяздисципліни«Безпека життєдіяльності, основи охорони праці» для студентівосвітньогоступеня „бакалавр" усіх спеціальностей та форм навчання / Укладачі : О. Я. Гурик,І. Б. Окіпний, В. С. Сенчишин, С. Ю. Мариненко, О. І. Король. Тернопіль:ТНТУ 70імені Івана Пулюя, 2025. 123 с. URL: http://elartu.tntu.edu.ua/handle/lib/48496

49. Грибан В. Г., Негодченко О. В. Г 82 Охорона праці. Навч. посіб.2-гевид. - К.: Центр учбової літератури, 2019. - 280 с.

ДОДАТКИ

На рисунку 2.6 показано загальну структуру інтерфейсу веб-додатку.

Інтерфейс умовно поділено на дві основні частини: навігаційну область та робочу область користувача.

Додаток А

Додаток Б - Код діаграми варіантів використання (Use Case)

Діаграма варіантів використання відображає основні сценарії взаємодії користувачів із системою аналізу якості програмного коду. Вона демонструє розподіл функціональності відповідно до ролей користувачів, де кожен актор взаємодіє із системою відповідно до своїх задач у процесі командної розробки.

```
@startuml
left to right direction

actor "Член команди розробки" as Dev
actor "Координатор якості коду" as QA
actor "Менеджер проєкту" as PM
actor "Адміністратор системи" as Admin

rectangle "Веб-додаток аналізу якості коду" {

  Dev --> (Перегляд показників якості коду)
  Dev --> (Аналіз змін у коді)
  Dev --> (Перегляд детальних метрик)

  QA --> (Перегляд узагальненої оцінки якості)
  QA --> (Аналіз динаміки змін)
  QA --> (Виявлення проблемних ділянок)
  QA --> (Аналіз внеску розробників)

  PM --> (Перегляд узагальненої оцінки якості)
  PM --> (Перегляд динаміки якості)

  Admin --> (Налаштування параметрів аналізу)
  Admin --> (Керування користувачами)

}
@enduml
```

Додаток Б - Код архітектури системи аналізу якості програмного коду

Архітектура програмного застосунку складається з рівня представлення, прикладної логіки, аналітичного рівня та рівня доступу до даних. Така структура забезпечує розподіл функціональності між компонентами системи та дозволяє реалізувати гнучкий підхід до обробки та аналізу даних.

```
@startuml
skinparam componentStyle rectangle
skinparam linetype ortho

actor "Користувач" as User

rectangle "Рівень представлення\n(Інтерфейс користувача)" as UI {
  [Веб-інтерфейс]
}

rectangle "Рівень прикладної логіки" as Logic {
  [Обробка запитів]
  [Управління процесом аналізу]
}

rectangle "Аналітичний рівень" as Analytics {
  [Обчислення метрик якості]
  [Аналіз змін коду]
  [Формування оцінок]
}

rectangle "Рівень доступу до даних" as Data {
  [Робота з базою даних]
}

database "База даних" as DB

' --- зв'язки ---
User --> UI

UI --> Logic

Logic --> Analytics
Logic --> Data

Analytics --> Data
Data --> DB

@enduml
```

Додаток В - Розширена логічна модель бази даних

Розширена логічна модель бази даних програмного рішення. Вона відображає основні сутності предметної області та зв’язки між ними, які забезпечують збереження та обробку інформації про процес аналізу якості програмного коду.

```
@startuml
hide circle
skinparam linetype ortho

entity "Користувач" as users {
  * id : int
  --
  name : string
  email : string
  role : string
}

entity "Проєкт" as projects {
  * id : int
  --
  name : string
  description : string
}

entity "Коміт" as commits {
  * id : int
  --
  date : datetime
}

entity "Метрика якості" as metrics {
  * id : int
  --
```

```
  name : string
  description : string
}

entity "Результат аналізу" as results {
  * id : int
  --
  value : float
}

entity "Внесок користувача" as contributions {
  * id : int
  --
  contribution_type : string
}

' --- зв’язки ---
users ||--o{ commits : створює
projects ||--o{ commits : містить

commits ||--o{ results : має
metrics ||--o{ results : використовується

users ||--o{ contributions : виконує
commits ||--o{ contributions : включає

@enduml
```

Фізична модель бази даних визначає реалізацію логічної структури у вигляді таблиць, їх полів та зв’язків у базі даних. Вона є безпосереднім відображенням сутностей, представлених у логічній моделі.

```
@startuml
hide circle
skinparam classAttributeIconSize 0

entity "users\n(Користувачі)" as users {
  * id : INT
  --
  name : VARCHAR
  email : VARCHAR
  role : VARCHAR
}

entity "projects\n(Проєкти)" as projects {
  * id : INT
  --
  name : VARCHAR
  description : TEXT
}

entity "versions\n(Версії коду)" as versions {
  * id : INT
  --
  project_id : INT (FK)
  version_number : VARCHAR
  created_at : DATETIME
}

entity "commits\n(Коміти)" as commits {
  * id : INT
  --
  project_id : INT (FK)
  user_id : INT (FK)
  message : TEXT
  date : DATETIME
}

entity "metrics\n(Метрики якості)" as metrics {
  * id : INT
  --
  name : VARCHAR
  description : TEXT
}
```

```
entity "analysis_results\n(Результати аналізу)" as results {
  * id : INT
  --
  commit_id : INT (FK)
  metric_id : INT (FK)
  value : FLOAT
}

entity "project_analysis\n(Аналіз проєкту)" as project_analysis {
  * id : INT
  --
  project_id : INT (FK)
  created_at : DATETIME
  summary_score : FLOAT
}

entity "user_contributions\n(Внесок користувачів)" as contributions
{
  * id : INT
  --
  user_id : INT (FK)
  commit_id : INT (FK)
  contribution_type : VARCHAR
}

' --- зв'язки ---
users ||--o{ commits
users ||--o{ contributions

projects ||--o{ commits
projects ||--o{ versions
projects ||--o{ project_analysis

versions ||--o{ commits

commits ||--o{ results
metrics ||--o{ results

commits ||--o{ contributions

@enduml
```

Додаток Д - UML-діаграма класів

Для наочного представлення структури програмної системи та взаємодії її компонентів далі побудовано UML-діаграму класів.

```
@startuml
skinparam classAttributeIconSize 0

class User {
  id : int
  name : string
  email : string
  role : string
}

class Project {
  id : int
  name : string
  description : string
}

class Version {
  id : int
  versionNumber : string
  createdAt : datetime
}

class Commit {
  id : int
  message : string
  date : datetime
}

class Metric {
  id : int
  name : string
  description : string
}

class AnalysisResult {
  id : int
  value : float
}
```

```
class ProjectAnalysis {
  id : int
  summaryScore : float
  createdAt : datetime
}

class Contribution {
  id : int
  contributionType : string
}

class AnalysisService {
  +calculateMetrics()
  +generateReport()
}

' --- зв'язки ---
User "1" -- "0..*" Commit
User "1" -- "0..*" Contribution

Project "1" -- "0..*" Commit
Project "1" -- "0..*" Version
Project "1" -- "0..*" ProjectAnalysis

Version "1" -- "0..*" Commit

Commit "1" -- "0..*" AnalysisResult
Metric "1" -- "0..*" AnalysisResult

Commit "1" -- "0..*" Contribution

ProjectAnalysis "1" -- "0..*" AnalysisResult

AnalysisService ..> Commit
AnalysisService ..> AnalysisResult
AnalysisService ..> Metric
AnalysisService ..> ProjectAnalysis

@enduml
```

Додаток Е - Діаграма діяльності процесу аналізу якості програмного коду

Для кращого розуміння послідовності виконання основних операцій у системі було побудовано діаграму діяльності, яка відображає процес аналізу якості програмного коду.

```
@startuml
start

:Завантаження даних про коміт;

:Перевірка коректності даних;

if (Дані коректні?) then (Так)

:Обчислення метрик якості;

:Формування результатів аналізу;

:Збереження результатів;

:Формування узагальненого звіту;

:Відображення результатів користувачу;

else (Ні)

:Повідомлення про помилку;

endif

stop
@enduml
```

ДОДАТОК Є - Фрагменти програмної реалізації вебзастосунку

Лістинг Є.1 – Приклад структури таблиці бази даних

Наведений фрагмент демонструє приклад створення таблиці projects, яка використовується для збереження інформації про програмні проєкти у базі даних вебзастосунку. Таблиця містить основні поля, необхідні для ідентифікації проєкту, збереження його назви, опису та дати створення.

```
CREATE TABLE projects (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    description TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Лістинг Є.2 – Приклад серверного API-запиту

Наведений фрагмент коду демонструє приклад реалізації серверного API-запиту для отримання списку програмних проєктів. Серверна частина обробляє HTTP-запит, звертається до бази даних та повертає результати у форматі JSON. Також передбачено базову обробку помилок у випадку некоректного виконання запиту.

```
app.get('/projects', async (req, res) => {
    try {
        const projects = await Project.findAll();
        res.json(projects);
    } catch (error) {
        res.status(500).json({
            message: 'Помилка отримання даних'
        });
    }
});
```

Лістинг Є.3 – Приклад компонента користувацького інтерфейсу

Наведений компонент використовується для відображення списку програмних проєктів у користувацькому інтерфейсі вебзастосунку. Компонент отримує дані у вигляді масиву об’єктів та формує структуроване представлення інформації для користувача. Використання компонентного підходу спрощує підтримку інтерфейсу та дозволяє повторно використовувати окремі елементи системи.

```
function ProjectList({ projects }) {
    return (
        <div className="project-list">

            <h2>Список проєктів</h2>

            {projects.map(project => (
                <div key={project.id}>
                    <p>{project.name}</p>
                </div>
            ))}

        </div>
    );
}
```

Лістинг Є.4 – Приклад запуску вебзастосунку

Наведені команди використовуються для встановлення необхідних залежностей та запуску вебзастосунку у локальному середовищі розробки. Використання пакетного менеджера npm дозволяє автоматизувати процес налаштування програмного середовища та спрощує розгортання вебзастосунку.

```
npm install
npm start
```

Міністерство освіти і науки України
Тернопільський національний технічний університет
імені Івана Пулюя
Маріборський університет (Словенія)
Технічний університет в Кошице (Словаччина)
Каунаський технологічний університет (Литва)
Львівський національний університет
імені Івана Франка
Гірничо-металургійна академія ім. Станіслава Сташиця (Польща)
Луцький національний технічний університет
Чернівецький національний університет
імені Юрія Федьковича
Вроцлавський економічний університет (Польща)
Університет технологій та економіки
імені Хелени Ходковської (Польща)
Донбаська державна машинобудівна академія

Студентське наукове товариство

IX МІЖНАРОДНА

студентська науково - технічна конференція

"ПРИРОДНИЧІ ТА ГУМАНІТАРНІ НАУКИ. АКТУАЛЬНІ ПИТАННЯ"

24-25 квітня 2026 р.

(збірник тез конференції)

Тернопіль 2026

УДК 004.41
Кузьмак Є.–ст. гр. СПс-41
Тернопільський національний технічний університет імені Івана Пулюя

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗОВАНОГО АНАЛІЗУ ЯКОСТІ ПРОГРАМНОГО КОДУ В БАГАТОКОРИСТУВАЦЬКИХ ПРОЄКТАХ

Науковий керівник: к.т.н., доцент Цуприк Г. Б.

Kuzmak Ye.
Ternopil Ivan Puluj National Technical University

DEVELOPMENT OF SOFTWARE FOR AUTOMATED ANALYSIS OF SOURCE CODE QUALITY IN MULTI-USER PROJECTS

Supervisor: PhD, Associate Professor H. B.Tsupryk

Ключові слова: інженерія програмного забезпечення, код, аналіз.
Keywords: Software Engineering, Code, Analysis.

При сучасних тенденціях розвитку галузі інженерії програмного забезпечення велика кількість програмних продуктів створюється саме командами фахівців. Тут над програмним проєктом одночасно можуть працювати декілька розробників з різним рівнем досвіду, а також стилем програмування. Саме тому питання забезпечення якості програмного коду набуває особливої актуальності, бо очевидно, що зростає складність самих проєктів, кількість поточних змін, ну і ризик виникнення помилок. Традиційні підходи до контролю якості програмного коду часто базуються на ручному аналізі та код-рев'ю. Тому існує висока ймовірність накопичення технічних помилок, ускладнення супроводу, зниження загальної ефективності командної розробки.

У зв'язку з цим актуальним є використання автоматизованих програмних засобів, які дозволяють здійснювати аналіз якості програмного коду, формувати звіти та узагальнюють інформацію. Як форма реалізації засобів, які забезпечуватимуть зручний доступ до результатів аналізу для різних категорій користувачів (розробників, тестувальників та керівників проєктів) є вебзастосунки. Отже, розробка веб-застосунку для аналізу якості програмного коду у командних програмних проєктах є актуальною задачею, яка відповідає сучасним тенденціям інформаційних технологій та потребам практичної діяльності в процесі повного циклу розробки програмних продуктів.

На практиці контроль якості програмного коду часто, окрім код-рев'ю, здійснюється за допомогою тестування та дотримання внутрішніх стандартів розробки. Але такі підходи часто суттєво залежать від людського фактору, бо існує проблема суб'єктивності оцінок або так званого людського фактору. Крім того, зі зростанням обсягів коду та складності програмних систем «ручний» аналіз стає доволі трудомістким і об'єктивно значно менш ефективним, особливо для команд.

Сучасні існуючі програмні засоби автоматизованого аналізу коду дозволяють частково вирішити перелічені проблеми. Зокрема, це роблять через використання статичних метрик, правил перевірки та формування звітів про стан кода. Однак більшість існуючих рішень орієнтовано або на окремі мови програмування, або на інтеграцію з конкретними середовищами розробки. Це суттєво ускладнює їх використання у різнорідних проєктах, коли працює команда розробників.

Окремою проблемою є представлення результатів аналізу у зручному для користувачів вигляді, оскільки часто отримані звіти є надто технічними та не дуже зрозумілими для керівників проєктів чи інших учасників команди, які не займаються безпосередньо програмуванням, що суттєво знижує їх практичну цінність.

Таким чином, існує реальна потреба у програмному рішенні, яке б поєднувало автоматизований аналіз якості програмного коду з можливістю наочного та зрозумілого представлення результатів для різних категорій користувачів. Отже, розробка веб-застосунка для аналізу якості програмного коду в роботі з великими проєктами, що потребують командної роботи спрямована саме на часткове вирішення зазначеної проблеми та підвищення ефективності процесів контролю якості програмного забезпечення.

Наукова новизна кваліфікаційної роботи полягає у підході до поєднання автоматизованого аналізу якості програмного коду з веб-орієнтованим представленням результатів для різних ролей користувачів у великих командних програмних проєктах.

В роботі я пропоную структурований спосіб узагальнення результатів аналізу коду, який дозволить представити технічні метрики у зрозумілому вигляді для всіх учасників коменди. На відміну від окремих існуючих інструментів аналізу коду, акцент в роботі я зробив не лише на виявленні технічних проблем, але й на організацію та збереження результатів аналізу з урахуванням історії перевірок в контексті командної розробки. Це дозволить використовувати результати аналізу не тільки для виправлення помилок, але й для оцінювання в часі загального стану якості коду.

Запропоноване програмне рішення може бути застосоване для комерційних проєктів для аналізу стану коду та отримання узагальнених зрозумілих показників якості. Результати роботи можуть бути використані розробниками та тестувальниками для своєчасного виявлення проблем, а керівниками проєктів, для отримання в зрозумілій формі інформації, яка значно спрощує прийняття рішень щодо подальшої роботи над проєктом, його розвитку або супроводу. Крім того, розроблені архітектурні та проєктні рішення можуть бути застосовні як основа для подальшого розширення функціональних можливостей системи або інтеграції з іншими програмними засобами.

Література:

1. Петров І. І., Сидоренко О. В. Оцінювання якості програмного коду за допомогою статичного аналізу [Електронний ресурс] // Одеська державна академія технічного регулювання та якості. – 2021. – Режим доступу: https://odatrya.org.ua/index.php/osatrq/article/view/346

2. Олянін, Д., Цуприк, Г. (2025) Transformer Neural Networks in Industry 4.0 / Д. Олянін, Г. Цуприк, Т. Говорущенко, О. Багрій-Заяць, І. Андрущак // Computer Information Technologies in Industry 4.0: proceedings of the 3rd International Workshop (CITI-2025), Ternopil, Ukraine, 11–12 June 2025. – Ternopil : Ternopil Ivan Puluj National Technical University, 2025 (Scopus) https://ceur-ws.org/Vol-4057/

3. Tsupryk, H., Olianin, D. (2025). Vydobuvannia danyh z tekstu vykorystovuiuchy transformerni neironni merezhi [Data extraction from text using Transformer Neural Networks]. Information Technology: Computer Science, Software Engineering and Cyber Security, 125–130, DOI: https://doi.org/10.32782/IT/2025-2-13

4. ОЛЯНІН D., & ЦУПРИК Н. (2025). Огляд ролі трансформерних нейроних мереж у видобувані інформації із неструктурованих даних. Measuring and computing devices in technological processes, 82(2), 360–364. https://doi.org/10.31891/2219-9365-2025-82-52