Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Метод формалізації вимог до програмного забезпечення на основі оцінки якості результатів опитування зацікавлених сторін
Назва теми

Рівень вищої освіти Другий (магістерський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ. 240161.01.03.ПЗ

Виконав студент 2 курсу, група ІПЗм-24-1 _____ Підпис Є.О. ГОРДІЄНКО Ініціали, прізвище

Керівник канд. пед. наук, доцент Науковий ступінь, звання _____ Підпис О.Г. ОНИШКО Ініціали, прізвище

Нормоконтролер канд. тех. наук, доцент _____ Підпис О.М. ЯШИНА Ініціали, прізвище

До захисту допускаю:
Завідувач кафедри інженерії програмного забезпечення _____ Підпис Л.П. БЕДРАТЮК Ініціали, прізвище

15 грудня 2025 р.

Хмельницький 2025

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Другий (магістерський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ІПЗ
Л. П. Бедратюк
01 09 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Гордієнку Євгенію Олеговичу
Прізвище, ім'я, по батькові студента

1. Тема проєкту (роботи) Метод формалізації вимог до програмного забезпечення на основі оцінки якості результатів опитування зацікавлених сторін

Керівник проєкту (роботи) канд. пед. наук, доцент Онишко О.Г.
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 25.08.2025 №65
2. Строк подання студентом проекту (роботи) на кафедру 01.12.2025 р.
3. Вихідні дані до проекту (роботи) Матеріали науково-дослідної практики
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
1 Аналіз предметної області та рішень з програмного забезпечення.
2 Удосконалення методу формалізації вимог до програмного забезпечення із використанням оцінювання якості результатів зацікавлених сторін
3 Архітектура програмної реалізації методу
4 Програмна реалізація системи для формалізації вимог до програмного забезпечення із використанням оцінювання якості результатів опитування зацікавлених сторін
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Презентаційні матеріали (слайди)

6. Консультанти розділів дипломного проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Антиплагіат	Форкун Ю.В., доцент	1.12.2025	15.12.2025
Нормоконтроль	Яшина О.М., доцент	30.11.2025	13.12.2025

7. Дата видачі завдання «01» вересня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1. Вивчення предметної області; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	20.10 - 26.09.2025	
2. Робота над розділом 1 кваліфікаційної роботи - аналіз предметної області та постановка завдання	20.10 - 26.09.2025	
3 Робота над розділом 2 кваліфікаційної роботи - визначення теоретичних засад	27.10 - 02.11.2025	
4. Робота над науковими публікаціями, статтями	03.11 - 09.11.2025	
5. Робота над розділом 3. Проектування архітектури системи для вирішення задачі, розробка вимог.	10.11 - 16.11.2025	
6 Робота над розділом 4 кваліфікаційної роботи - формалізація, оцінювання та порівняльний аналіз запропонованого підходу	17.11 - 23.11.2025	
7 Попередній захист кваліфікаційної роботи	24.11 - 30.11.2025	
8 Узгодження постановки задачі, отриманих результатів та висновків; оформлення пояснювальної записки та графічних матеріалів згідно вимог чинних стандартів	01.12-07.12.2025	
9 Перевірка роботи на наявність плагіату: нормоконтроль; брошурування пояснювальної записки; підготовка супровідних документів	08.12 - 14.12.2025	
10 Підготовка до захисту кваліфікаційної роботи	З 15.12.2025	

Студент _____ Є.О. ГОРДІЄНКО
Підпис Ініціали, прізвище

Керівник проєкту (роботи) _____ О.Г. ОНИШКО
Підпис Ініціали, прізвище

РЕФЕРАТ

Тема дипломної роботи: «Метод формалізації вимог до програмного забезпечення на основі оцінки якості результатів опитування зацікавлених сторін».

Автор роботи: Гордієнко Євгеній Олегович.

Керівник роботи: Онишко Оксана Григорівна.

Пояснювальна записка: 85 с., 18 рис., 2 дод., 25 джерел.

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ЯКІСТЬ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ФОРМАЛІЗАЦІЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.

Мета роботи – удосконалення методу формалізації вимог до програмного забезпечення із використанням оцінювання якості результатів опитування стейкголдерів (зацікавлених сторін).

Об’єктом дослідження є процес формалізації вимог до програмного забезпечення на етапі аналізу вимог до розроблюваного програмного продукту.

Предметом дослідження є методи формалізації вимог до програмного забезпечення на етапі аналізу вимог до розроблюваного програмного продукту.

Новизна:

1. Удосконалення методу формалізації вимог до програмного забезпечення із використанням оцінювання якості результатів опитування зацікавлених сторін.

Щоб здійснити написання магістерської роботи потрібно виконати ряд таких завдань:

1. Провести детальний аналіз предметної області щодо можливостей визначення вимог до програмного забезпечення із використанням певних методів, засобів та існуючих технологій.
2. Здійснити аналітичний огляд існуючих методів формалізації вимог до програмного забезпечення.
3. Удосконалити метод формалізації вимог до програмного забезпечення із використанням оцінювання якості результатів опитування зацікавлених сторін.

Практичне значення отриманих результатів полягає в тому, що її результати та теоретичні напрацювання можуть використовуватись у роботі розробників програмного забезпечення під час аналітичного процесу щодо визначення вимог та формулювання технічного завдання.

Результати даної магістерської роботи можуть бути використані у подальших наукових дослідженнях, а також аналітиками у сфері IT-розробки під час проектування програмного забезпечення.

05.12.2025

ABSTRACT

Master’s thesis: «Method for formalising software requirements based on the assessment of the quality of stakeholder survey results».

Author: Gordienko Yevgeniy.

Head of work: Onyshko Oksana.

Master’s thesis consists of: 85 pages of the general text, 18 graphics, 2 supplements, 25 literature sources.

SOFTWARE REQUIREMENTS, ANALYSIS OF SOFTWARE REQUIREMENTS, SOFTWARE QUALITY, FORMALISATION OF SOFTWARE REQUIREMENTS

The aim of the work is to improve the method of formalising software requirements using the assessment of the quality of stakeholder survey results.

The object of the study is the process of formalising software requirements at the stage of analysing requirements for the software product being developed.

The subject of the study is methods for formalising software requirements at the stage of analysing requirements for the software product being developed.

Novelty:

1. Improving the method of formalising software requirements using the assessment of the quality of stakeholder survey results.

To write a master's thesis, a number of tasks must be completed:

1. Conduct a detailed analysis of the subject area regarding the possibilities of determining software requirements using specific methods, tools, and existing technologies.

2. Conduct an analytical review of existing methods for formalising software requirements.

3. Improve the method of formalising software requirements using the assessment of the quality of stakeholder survey results.

The practical significance of the results obtained lies in the fact that its results and theoretical developments can be used in the work of software developers during the analytical process of determining requirements and formulating technical tasks.

The results of this master's thesis can be used in further scientific research, as well as by analysts in the field of IT development when designing software.

05.12.2025

ЗМІСТ

ВСТУП

Актуальність дослідження. Допоміжним засобом для систематичної розробки програмного забезпечення та вдосконалення процесів є приведення до стандартизації інженерії програмного забезпечення. Через неформальне представлення знань з інженерії програмного забезпечення аналітичний документ із вимогами до розроблюваного програмного забезпечення може бути непослідовним, неоднозначним та неповним. Більш формальне представлення дозволяє подолати вказані недоліки. Загалом, моделювання процесів розробки програмного забезпечення – це напрямок у програмній інженерії, який спрямований на забезпечення підтримки для фіксації процесів розробки програмного забезпечення. Саме тому використання методів формалізації вимог до програмного забезпечення є актуальним завданням під час визначення таких вимог в життєвому циклі програмного продукту, зокрема й на основі оцінки якості результатів опитування зацікавлених сторін.

Загалом неявні знання зацікавлених сторін є ключовим етапом виявлення вимог, а отже, і розробки програмного забезпечення загалом. Разом з тим такий критичний елемент має значний вплив на визначення результату та якості вимог, а отже, і цілісної розробки. Через свою неявну природу, він є прихованим і глибоко захованим у свідомості зацікавлених сторін, тому його надзвичайно важко сформулювати та передати, а також ще важче виявити та використати. Крім того, в літературі повідомляється, що існує дефіцит доступних теоретичних розробок і рішень для вирішення цієї проблеми, що створює ключову та постійну проблему в її успішному досягненні, функціональному використанні, теоретичному розумінні та синтезі, а також успішному зборі. У даній роботі представлена теоретична структура управління знаннями для отримання неявних знань, профіль зацікавленої сторони, з метою внеску в корпус знань. Структура пропонує теоретичне бачення отримання неявних знань зацікавленими сторонами як окремими особами та концептуальне бачення його

застосування в контексті спеціально розробленої моделі процесу співбесіди з виявлення вимог. З огляду на цей контекст, ця структура пропонує цілісне концептуальне бачення рішення, включаючи аналіз пом'якшувальних факторів для отримання неявних знань з урахуванням інтерв'ю, теоретичну основу, синтез та отримання неявних знань зацікавлених сторін як окремих осіб, інтеграцію бачення в модель процесу, специфічну для інтерв'ю та концептуальну оцінку показників якості результатів процесу інтерв'ю за допомогою програмного засобу.

Мета роботи – удосконалення методу формалізації вимог до програмного забезпечення із використанням оцінювання якості результатів опитування стейкголдерів.

Об’єктом дослідження є процес формалізації вимог до програмного забезпечення на етапі аналізу вимог до розроблюваного програмного продукту.

Предметом дослідження є методи формалізації вимог до програмного забезпечення на етапі аналізу вимог до розроблюваного програмного продукту.

Новизна:

1. Удосконалення методу формалізації вимог до програмного забезпечення із використанням оцінювання якості результатів опитування зацікавлених сторін.

Щоб здійснити написання магістерської роботи потрібно виконати ряд таких завдань:

4. Провести детальний аналіз предметної області щодо можливостей визначення вимог до програмного забезпечення із використанням певних методів, засобів та існуючих технологій.
5. Здійснити аналітичний огляд існуючих методів формалізації вимог до програмного забезпечення.
6. Удосконалити метод формалізації вимог до програмного забезпечення із використанням оцінювання якості результатів опитування зацікавлених сторін.

Практичне значення отриманих результатів полягає в тому, що її результати та теоретичні напрацювання можуть використовуватись у роботі розробників програмного забезпечення під час аналітичного процесу щодо визначення вимог та формулювання технічного завдання.

Результати даної магістерської роботи можуть бути використані у подальших наукових дослідженнях, а також аналітиками у сфері IT-розробки під час проектування програмного забезпечення.

Відповідно до теми кваліфікаційної роботи опубліковані тези «Аналіз інструментів та засобів формалізації вимог при проектуванні та розробці програмного забезпечення» на конференції «Актуальні проблеми комп’ютерних наук (АПКН-2025).

1 ТЕОРЕТИЧНИЙ ВИКЛАД ДОСЛІДЖУВАНОЇ ПРОБЛЕМИ

1.1 Аналіз предметної області

Визначення та розробка вимог – це найперший етап життя розробника програмного забезпечення. Інженерія вимог до програмного забезпечення визначається як спеціалізований набір комплексна діяльність з виявлення, застосування повномасштабного аналізу для подальшого розвитку вимоги, документування як специфікації та, нарешті, перевірка задокументованого вимоги стосовно реальних потреб користувача. Крім того, і як невід'ємна частина цього розвитку, в сукупності виявлення вимог визначається як точний процес, за якого вимоги до програмного забезпечення розкриваються або виявляються, виявляються на поверхню, визначаються, витягуються, зібрані та встановлені.

Саїдіан та Дейл визначили ключові елементи, що визначають якість атрибутів та результат процесу виявлення, оскільки ключові залучені гравці, а саме користувачі та розробники. Вони пояснюють, що з точки зору користувачів, якісне виявлення інформації забезпечує їх здатністю краще розуміти свої межі та потреби, щоб вони були більш ефективно зважувати свої варіанти та їхні відповідні наслідки. З точки зору розробників, виявлення якості дає чітке уявлення про специфікація поточних викликів з точки зору високого рівня. Це також підкреслює, що справжні проблеми, які зараз розглядаються, – це ті, які насправді вирішуються, і що запропоновані рішення є здійсненними. А найважливішим аспектом процесу виявлення якості є процес, у якому розробник і користувач мають взаємний розуміння та спільна перспектива й бачення поточної проблеми.

IEEE визначає вимогу наступним чином з різних точок зору:

1. умова або здатність, необхідна користувачеві для вирішення проблеми або досягнення певного результату мета;

2.умова або здатність, якій має відповідати або володіти система або системний компонент для задоволення вимог контракту, стандарту, специфікації чи інших офіційно накладений документ; документоване відображення стану або здатність, як у визначенні першому чи другому пункті.

Знання зацікавлених сторін є ключовим сегментом у процесі визначення вимог до програмного забезпечення. Успіх розробки програмного проєкту або продукту визначається насамперед точністю виявлення та задоволенням потреб зацікавлених сторін, що, своєю чергою, залежить від ефективного збору та врахування їх неявних (мовчазних) знань. Ця теза широко підтверджена та обговорюється в науковій літературі.

Неявні знання ЗС не існують ізольовано, а є є центральною концепцією, нерозривно пов'язаною з процесами виявлення вимог (Requirements Elicitation, RE), інженерії вимог (Requirements Engineering, RE) та життєвим циклом розробки програмного забезпечення (Software Development Life Cycle, SDLC).

Неявні знання розташовані в центрі критичної та нестабільної фази інженерії вимог і їх неналежне врахування має значний потенціал для спричинення невдачі всього проєкту або розробки продукту. Цей вид знань тісно корелює з практичними посібниками та механізмами інженерії вимог, зокрема з виявленням вимог та його важелями. Розташування неявних знань у ядрі процесу виявлення вимог створює значну кількість викликів та перешкод. Пошук методів виявлення неявних знань зацікавлених сторін та їх формалізації є ключовою та бажаною метою, що багаторазово підкреслюється в науковій спільноті.

Неявні знання зацікавлених сторін вважаються одним із найважливіших елементів, що зумовлює успіх або невдачу процесу виявлення вимог, його результату та загалом розробки проєкту чи програмного продукту.

Проблеми, пов'язані з опрацюванням неявних знань зацікавлених сторін, традиційно розглядаються в рамках управління знаннями. Однак, невирішеними залишаються питання, що стосуються індивідуалізованої теоретизації неявного

знання людей як окремих суб'єктів, а не просто як предмет огляду літератури. Існує широкий заклик до розробки рішень та інновацій для подолання цих ключових невирішених питань. Це створює дослідницьку прогалину та необхідність розробки й впровадження відповідного рішення.

Впровадження структурованих циклів зворотного зв'язку від запитів користувачів може призвести до суттєвого покращення продуктивності продукту. Наприклад, компанії, які аналізують взаємодію з клієнтами, повідомляють про 20% збільшення задоволеності продуктом, як було зазначено в опитуванні Асоціації клієнтського досвіду 2024 року [3]. Регулярний збір відгуків користувачів одразу після взаємодії зі службою підтримки може точно визначити повторювані проблеми, дозволяючи командам ефективно їх вирішувати.

Використання інструментів аналізу даних для категоризації запитів надає практичну інформацію. У галузевому звіті за 2023 рік було зазначено, що компанії, які використовують аналітику, покращили швидкість реагування, мінімізуючи час простою.

Регулярно заплановані зустрічі після завершення, присвячені невирішеним проблемам із сесій підтримки, сприяють культурі постійного вдосконалення. Згідно зі звітом Agile Software Development за 2024 рік, понад 60% Agile-команд, які включили такі сесії зворотного зв'язку, відзначили помітне зниження повторюваності проблем протягом кварталу.

Заохочення кінцевих користувачів до участі в тестуванні продукту перед основними релізами може бути корисним. Дослідження DevOps Research and Assessment показало, що компанії, які впроваджують ранні цикли зворотного зв'язку з користувачами, стикаються з меншою кількістю критичних проблем після запуску, що призводить до більш спрощеного шляху виходу на ринок.

Використання інструменту зворотного зв'язку може автоматизувати збір пропозицій та критики користувачів, забезпечуючи відкритість каналів для

постійного діалогу. Рівень залученості покращується завдяки інтеграції зручних платформ зворотного зв'язку. Аналіз цих даних не лише вирішує поточні проблеми, але й допомагає в майбутніх оновленнях, узгоджуючи розробку продукту з очікуваннями користувачів.

Впровадження цих практик не лише вирішує нагальні скарги, але й створює адаптивну екосистему, стійку до майбутніх викликів. Впровадження ефективних механізмів зворотного зв'язку не просто корисне; це стратегічна необхідність у сучасному середовищі.

1.2 Аналіз існуючих рішень

Формалізація вимог до програмного забезпечення на основі відповідей зацікавлених сторін (ЗС) є ключовою проблемою в інженерії вимог (Requirements Engineering, RE). Дослідницькі статті та методології пропонують низку рішень для перетворення неформальних, часто виражених природною мовою (Natural Language, NL), відповідей зацікавлених сторін на формальні, точні та однозначні специфікації. Загалом виділяють такі напрямки: використання формальних методів та мов моделювання, методологія KAOS, трансформація через уточнення.

Напрямок, що використовує формальні методи та мови моделювання зосереджений на застосуванні математичної строгості для підвищення точності вимог, отриманих від ЗС. При використанні формальних мов моделювання пропонуються спеціалізовані мови моделювання, які можуть фіксувати як функціональні, так і нефункціональні вимоги, а також зв'язки між ними. Це дозволяє трансформувати неформальні вимоги ЗС у формальні специфікації за допомогою систематичних процесів.

Методологія KAOS полягає у тому, що в рамках цієї методології цілі, виявлені у ЗС, формалізуються за допомогою лінійної темпоральної логіки

(LTL). Ці формалізовані цілі потім уточнюються до підцілей і операціоналізуються як специфікації системних операцій (з перед-, пост- та тригерними умовами) відповідно до набору формальних правил виведення.

Трансформація через уточнення (Refinement) полягає у тому, що однією з головних ідей є процес поступового уточнення неформальних вимог ЗС до формальної, узгодженої та вимірюваної специфікації, яка може бути передана розробникам.

2. Застосування методів обробки природної мови (NLP). Цей підхід вирішує проблему обробки великої кількості неструктурованих відповідей зацікавлених сторін, виражених природною мовою та включає такі методології: автоматичний збір та агрегація, підвищення точності.

При автоматичному зборі та агрегації пропонуються ефективні, автоматичні підходи до збору вимог із відповідей різних зацікавлених сторін на конкретне питання. Методи NLP використовуються для вимірювання подібності відповідей, щоб вибрати той варіант, який найкраще представляє думку більшості зацікавлених сторін.

Хоча природна мова сприяє зрозумілості для всіх зацікавлених сторін, їй бракує точності. NLP допомагає в аналізі тексту, щоб зменшити двозначність і нечіткість перед етапом формалізації.

3. Робочий процес затвердження вимог (Requirements Approval Workflow) зосереджується на організаційному та ітеративному аспекті, щоб гарантувати, що формалізовані вимоги відповідають очікуванням зацікавлених сторін. Такий метод складається із трьох етапів: створення робочого процесу, створення циклу зворотного зв'язку, зменшення ризиків.

Створення робочого процесу під час якого використовується процес затвердження, де визначені рецензенти, тобто зовнішні зацікавлених сторін та учасники, наприклад, співавтори, як правило, внутрішні, що переглядають та погоджують або відхиляють проєкт вимоги.

На етапі циклу зворотного зв'язку вимоги документуються, потім валідуються та верифікуються разом із зацікавленими сторонами для забезпечення їх точного представлення потреб. Це ітеративний процес, який передбачає постійне уточнення та оновлення вимог на основі зворотного зв'язку від зацікавлених сторін.

Процес зменшення ризиків сприяє активній участі зацікавлених сторін, покращує якість вимог, зменшує ризики відхилень обсягу проєкту (scope creep) та подальших переробок.

Ці підходи в сукупності забезпечують як методологічну строгість (формальні методи), так і практичну ефективність (NLP та робочі процеси) у процесі перетворення відповідей ЗС на функціональні та нефункціональні вимоги до ПЗ.

Загалом, дослідники виділяють декілька інструментів із доступністю поза межами проектів. Також можна виокремити ще один засіб, що доступний як частина набору інструментів COMPASS [1], який спрощує формальну специфікацію вимог у контексті моделей AADL [2].

У [3] автори представили інструмент під назвою DODT, розроблений у рамках проекту CESAR [4]. DODT дозволяє проекційне редагування та типізацію вимог на основі шаблонного синтаксису (Requirements Specification Language - RSL), атрибутної онтології та предметно-специфічної онтології. Були реалізовані перевірки валідації на основі онтологій, які дозволяють виявляти суперечності шляхом попарного порівняння вимог, іменників, які не визначені в онтології тощо. Нарешті, підтримується мова шаблонів властивостей з формальною семантикою. Хоча DODT не реалізує точного зв'язку шаблонів зі шаблонами властивостей, однак авторами надаються пропозиції щодо шаблонів, які можуть бути придатними для опису заданої вимоги.

Інструмент EARS-CTRL [5] був введений для забезпечення шляхом побудови коректності вимог, написаних з використанням шаблонів Easy Approach to Requirements Syntax (EARS). Інструмент перевіряє, чи можна

синтезувати контролер з набору вимог. Якщо контролер неможливо синтезувати, можливо, існують конфліктуючі вимоги. EARS-CTRL дозволяє редагувати проекційні вимоги на основі глосарію, визначеного для області синтезу контролерів. Вимоги аналізуються як формули LTL. Ефективність аналізу залежить від визначеної користувачем семантичної інформації (наприклад, простих предикатів) для заданого глосарію. Нарешті, синтез моделі обмежується фрагментом LTL, який включає універсальний квантифікатор шляху, оператор next-step та слабкий часовий оператор until.

Інструмент RERD [9] підтримує специфікацію вимог та виведення властивостей на основі шаблонів та шаблонів властивостей, а також поступове конструювання систем для реалізації похідних властивостей за допомогою підходу проектування знизу вгору. Під час редагування вимог користувач взаємодіє з базовими онтологіями, запитуючи їх та отримуючи доступ до екземплярів їхніх класів для пошуку відповідних термінів. Таким чином, словник шаблонних атрибутів обмежений термінами, які можна ідентифікувати в рамках предметно-специфічних онтологій та онтологій, специфічних для проектованої системи. Зв'язки шаблонних атрибутів відіграють ключову роль у семантичному аналізі та виведенні властивостей, де кожен шаблонний шаблон пов'язаний із заздалегідь визначеними шаблонами властивостей, які можуть формально його фіксувати. Для забезпечення дотримання властивостей користувач може вибрати один із доступних шаблонів архітектури та параметризувати їх, вибираючи компоненти з поступово побудованої моделі BIP системи. Потім необхідно перевірити відсутність взаємоблокувань у результуючій моделі BIP, що пропонується зробити за допомогою інструменту D-Finder. Для властивостей, які неможливо забезпечити за допомогою існуючих шаблонів архітектури, користувачеві доведеться використовувати зовнішні інструменти, такі як перевірка моделей nuXmv. Щодо підходу на основі COMPASS, у [1] автори представили інший підхід, згідно з яким формальні властивості не створюються з шаблонів шляхом заміни їхніх заповнювачів

станами/подіями, пов'язаними з атрибутами, визначеними шаблоном, а шляхом присвоєння значень атрибутам системної моделі. Більш конкретно, було введено набір властивостей, які дозволяють пов'язувати значення з певними елементами моделі AADL. Потім було визначено таксономію формальних властивостей, кожна з яких виражається значеннями, пов'язаними з відповідними властивостями AADL, або ж за допомогою певних структур у моделі AADL (наприклад, підкомпоненти та портові з'єднання). Формальна семантика системних властивостей спирається на поведінкову семантику мови SLIM, тоді як логіка, що використовується для визначення більшості з них, є варіантом метричної часової логіки (MTL). Хоча таксономія формальних властивостей порівняно обмежена у виразності, вона повністю усуває необхідність вибору шаблону та винаходження екземпляра. Формальні властивості можна аналізувати на узгодженість між різними рівнями абстракції системної моделі або використовувати як припущення/гарантії компонентів для проектування на основі контрактів. Для останньої мети COMPASS вимагає додаткової специфікації уточнення контракту, яке пов'язує контракт з контрактами підкомпонентів. Це дозволяє виконувати різні аналізи, такі як узгодженість або наслідки для будь-якої підмножини властивостей контракту, щоб перевірити правильність уточнень контракту, а також звужити контракт таким чином, щоб уточнення залишалося чинним.

Ефективність специфікацій вимог на основі онтологій та шаблонів залежить від якості предметно-орієнтованих онтологій та виразності мов шаблонів та їхнього потенціалу усунення неоднозначності.

Щодо першого питання, для проектування належних предметно-орієнтованих онтологій потрібно залучити інженерів вимог, інженерів онтологій та експертів з предметної області, які відповідають за проектування системи. Такий процес вимагає збору знань з попередніх проектів (наприклад, документів специфікацій вимог, документів архітектурного проектування), визначення важливих концепцій, які не є специфічними для цих проектів, а потім їх

класифікації в абстрактні категорії (ідентифікована функціональність, обмін інформацією між системами тощо). Потім категорії повинні бути призначені шаблонним атрибутам з концептуальної моделі, а їхні взаємозв'язки повинні бути онтологічно визначені. Також потрібно отримати неявні знання для задачі проектування, що необхідно, наприклад, для того, щоб зробити висновок, що певні події або діапазони даних, які синтаксично відповідають моделі шаблонних атрибутів, не є семантично релевантними. Такі знання можуть походити, наприклад, із законів фізики або інших аспектів системної інженерії. З точки зору онтологічної інженерії, необхідно прийняти адекватні якісні (згуртованість, адаптивність тощо) та кількісні (зв'язок, обчислювальна ефективність тощо) критерії, які допомагають виявляти помилки та неефективність щодо складності моделювання та розміру онтологій.

Додатковою проблемою є організація співпраці чи відповідальності різних відділів/зацікавлених сторін, які можуть бути залучені до різних онтологій, що у великих системних проектах може включати, наприклад, онтології предметної області, спільні для кількох зацікавлених сторін.

Виразність мов шаблонів важлива для можливості формалізувати всі системні вимоги, які потребують перевірки. Значною мірою виразність залежить від набору сполучних слів, що підтримуються синтаксисом мови шаблонів, та їхнього значення. Обмежений набір сполучників обмежує виразність мови, тоді як більш широкий набір сполучників може зробити неможливим повне уникнення неоднозначності в синтаксисі та семантиці. Для покращення виразності може бути навіть необхідно надати кілька значень сполучному слову, які потрібно буде розрізняти за позицією слова в реченні. У будь-якому випадку, синтаксис мов шаблонів має бути розроблений з урахуванням розширюваності.

Зусилля, необхідні для застосування будь-якого з розглянутих процесів чи інструментів у промисловому проекті, та їхній потенціал масштабованості створюють важливі проблеми. Більшість з них були застосовані до тематичних досліджень з перевірки концепції на основі підмножин реальних вимог,

наприклад, від космічних систем, але жоден з них не був застосований до повного набору вимог реального промислового проекту. Для такого масштабу проекту проблеми, які необхідно вирішити, включають документування зв'язку між специфікаціями вимог на основі онтологій та базовими вимогами, а також життєвим циклом онтологій у рамках проекту та між проектами, покращення зручності використання шляхом підвищення рівня технологічної готовності інструментів, особливо коли деякі властивості неможливо перевірити, і необхідно визначити відповідну підмодель для корекції.

Отже, специфікація вимог може базуватися на онтологіях предметної області, які використовуються для виявлення відсутньої інформації та усунення невідповідностей і випадків недостатньої специфікації. Виведення властивостей залежить від формальної моделі системи, яка може бути поступово побудована за допомогою компонентного процесу проектування.

Презентація цих розробок дозволяє прокоментувати їхні сильні та слабкі сторони. Також було очевидно, що важливі обмеження все ще потребують вирішення, а також відкриті проблеми, пов'язані із застосуванням такого підходу в проектах промислового масштабу. Окрім цих перспектив, існують відкриті проблеми для майбутніх досліджень щодо підвищення рівня автоматизації формалізації та валідації вимог, а також розширення застосування типів позафункціональних вимог або властивостей.

1.3 Огляд методів вирішення проблеми

Формалізація вимог до програмного забезпечення забезпечує чіткість, правильність і перевірюваність, а також вимагає формальних мов специфікації, логіки та методів верифікації, таких як доведення теорем і перевірка моделей, щоб гарантувати правильність розроблюваної програмної системи. Як і в усіх

інших галузях, розвиток великих мовних моделей відкрив світ можливостей, де ми можемо використовувати їхню потужність для генерації формальних вимог та супутніх специфікацій.

Аналітичний огляд методів формалізації вимог до програмного забезпечення демонструє широкий спектр підходів, що варіюються від структурованої природної мови до суворої математичної логіки, де вибір конкретного інструменту залежить від критичності системи та необхідного рівня точності. Цей континуум можна умовно поділити на методи, орієнтовані на людину, графічні нотації та формальні математичні методи.

На одному полюсі спектра знаходяться методи, що базуються на структурованій природній мові, які є стандартом для гнучких методологій розробки. Найпоширенішим прикладом тут є історії користувачів (User Stories). Цей метод фокусується на стислому описі потреби з погляду кінцевого користувача за шаблоном *Як [роль], я хочу [дія], щоб [цінність]*. Наприклад, у системі електронної комерції вимога може звучати так: *Як зареєстрований покупець, я хочу мати можливість повторити попереднє замовлення одним кліком, щоб заощадити час на пошук товарів.* Перевагою цього методу є висока зрозумілість для бізнесу, проте він часто страждає від неоднозначності, якщо не доповнений критеріями приймання.

Для більш детального опису поведінки системи використовуються варіанти використання. Цей метод формалізує вимогу через сценарій взаємодії актора (користувача) та системи. Наприклад, для функції *Зняття готівки в банкоматі* описується основний потік подій: користувач вставляє картку, система запитує PIN, користувач вводить код, система перевіряє баланс і видає гроші. Критично важливою особливістю даного методу є обов'язковий опис альтернативних потоків, наприклад, сценарію, коли користувач ввів неправильний PIN тричі або коли в банкоматі закінчилися купюри. Це дозволяє формалізувати обробку помилок на ранньому етапі.

Наступний рівень формалізації представлений напівформальними графічними нотаціями, які дозволяють візуалізувати структуру та поведінку системи, зменшуючи ризик різночитань тексту, де стандартом виступає UML (Unified Modeling Language). Для формалізації статичних вимог до даних часто використовують діаграми класів. Наприклад, моделюючи університетську систему, можна графічно зобразити клас *Студент* та клас *Курс*, з'єднавши їх зв'язком *багато-до-багато*, що чітко вказує розробнику на необхідність створення проміжної таблиці в базі даних. Для динамічних вимог застосовують діаграми послідовності, де, наприклад, процес онлайн-оплати буде складатись із таких дій: діаграма покаже вертикальні лінії життя для *Користувача*, *Вебінтерфейсу*, *Сервера* та *Платіжного шлюзу*, а стрілки між ними чітко визначать порядок викликів API та передачу даних у часі.

Для формалізації бізнес-процесів, які програмне забезпечення має автоматизувати, широко застосовується нотація BPMN (Business Process Model and Notation). Вона дозволяє описати наскрізний процес, що проходить через різні відділи або системи. Наприклад, процес *Узгодження відпустки* можна зобразити у вигляді діаграми з доріжками (swimlanes): співробітник ініціює запит, система автоматично перевіряє залишок днів відпустки (шлюз прийняття рішень), керівник отримує повідомлення і затверджує запит, після чого бухгалтерія отримує наказ. Така візуалізація робить вимогу прозорою як для розробників, так і для менеджерів.

Найвищий рівень точності забезпечують суворі формальні (математичні) методи, які застосовуються у критично важливих системах (авіація, медицина, фінанси), де ціна помилки є неприпустимо високою. Одним із таких методів є використання скінченних автоматів (Finite State Machines). Цей підхід ідеально підходить для опису систем, які можуть перебувати лише в чітко визначених станах. Наприклад, турнікет у метро, що він має два стани *Заблоковано* і Розблоковано. Формальний опис визначає, що перехід зі стану *Заблоковано* у *Розблоковано* можливий виключно при події *Оплата пройшла успішно*, а

зворотний перехід - при події *Користувач пройшов.* Це виключає будь-яку невизначеність поведінки пристрою.

Ще більш глибоку формалізацію пропонують методи, засновані на теорії множин та логіці предикатів, такі як Z-нотація або B-Method. У цьому випадку вимоги записуються мовою математичних формул. Наприклад, для системи керування складом хімічних речовин можна математично описати будь-який інваріант. Математичний доказ гарантує, що програма, написана за такою специфікацією, ніколи не допустить небезпечного стану, чого неможливо досягти звичайним тестуванням.

Окремо варто виділити сучасний підхід так званих виконуваних специфікацій, зокрема в рамках BDD (Behavior-Driven Development) з використанням мови Gherkin. Цей метод намагається поєднати зрозумілість природної мови з точністю коду. Вимоги записуються у форматі *Given / When / Then*. Унікальність цього методу полягає в тому, що такий текст автоматично перетворюється на автотест, який перевіряє роботу реального коду, фактично стираючи межу між вимогою, тестом та документацією.

Таким чином, вибір методу формалізації є компромісом між вартістю створення специфікації та ризиком виникнення дефектів: для стартапу достатньо User Stories, для банківського процесингу необхідні UML та BPMN, а для системи керування кардіостимулятором обов'язковим є застосування формальних математичних методів.

Математичні методи формалізації вимог, часто об’єднані терміном формальні методи, являють собою найбільш суворий підхід до специфікації програмного забезпечення, де поведінка системи описується не словами, а за допомогою математичних моделей, заснованих на теорії множин, логіці предикатів, алгебрі та теорії графів. Використання цих методів дозволяє не просто описати, як система має працювати, а математично довести коректність, повноту та безпеку вимог ще до написання першого рядка коду.

Одним із фундаментальних підходів є використання теорії множин та логіки першого порядку, яскравими представниками якого є Z-нотація (Z-notation) та B-Method. У цьому підході стан програмної системи моделюється як сукупність математичних множин та відношень між ними. Наприклад, база даних користувачів описується не як таблиця, а як абстрактна множина ідентифікаторів, а процес входу в систему - як математична операція, що змінює стан системи. Специфікація складається з так званих схем, які містять декларації змінних та предикати (логічні умови). Ключовим поняттям тут є інваріант стану - умова, яка повинна залишатися істинною завжди.

Для моделювання систем, що мають чітко виражену поведінку залежно від історії подій, застосовується теорія автоматів, зокрема скінченні автомати. Цей метод розглядає систему як набір дискретних станів і переходів між ними, які ініціюються вхідними подіями. Математичний опис автомата включає п'ять об'єктів: множину станів, початковий стан, вхідний алфавіт, функцію переходів та множину кінцевих станів. Формалізація вимог через автомати дозволяє виявити такі критичні помилки, як мертві стани (deadlocks), з яких система не може вийти, або недосяжні стани, які ніколи не будуть активовані. Це критично важливо для проєктування контролерів обладнання, протоколів зв'язку або логіки інтерфейсів.

У випадках, коли програмне забезпечення повинно виконувати багато процесів одночасно (паралельні та розподілені системи), для формалізації використовують мережі Петрі. Це математичний апарат, що поєднує теорію графів та лінійну алгебру. Структура мережі складається з місць (станів), переходів (подій) та фішок (токенів), що переміщуються між місцями. Мережі Петрі дозволяють формально описати та проаналізувати явища конкуренції за ресурси, синхронізації процесів та взаємного блокування.

Для специфікації систем реального часу та критичних до безпеки систем широко застосовується темпоральна логіка. Класична логіка оперує поняттями істина або хибність, тоді як темпоральна логіка додає вимір часу, дозволяючи

формулювати твердження про майбутнє. Вона використовує спеціальні оператори, такі як: завжди), колись у майбутньому, у наступний момент часу, доки.

Нарешті, алгебраїчні специфікації (наприклад, мови CASL або Larch) фокусуються на описі типів даних через їхні властивості, а не реалізацію. Тип даних визначається набором операцій та аксіомами, що описують відношення між цими операціями. Класичним прикладом є специфікація стеку, де аксіома визначає поведінку структури даних суто математично, абстрагуючись від того, як вона буде закодована - через масив чи зв'язний список. Цей метод дозволяє розробникам створювати програмні модулі, які гарантовано відповідають інтерфейсним контрактам, забезпечуючи високу надійність компонентної архітектури.

1.4 Постановка задачі

Для досягнення цілей наукового дослідження було встановлено мету дослідження, його об’єкт та предмет.

Мета роботи – удосконалення методу формалізації вимог до програмного забезпечення із використанням оцінювання якості результатів опитування стейкголдерів.

Об’єктом дослідження є формалізації вимог до програмного забезпечення на етапі аналізу вимог до розроблюваного програмного продукту.

Предметом дослідження є методи формалізації вимог до програмного забезпечення на етапі аналізу вимог до розроблюваного програмного продукту.

Новизна:

1. Удосконалення методу формалізації вимог до програмного забезпечення із використанням оцінювання якості результатів опитування стейкголдерів.

Щоб здійснити написання магістерської роботи потрібно виконати ряд таких завдань:

7. Провести детальний аналіз предметної області щодо можливостей визначення вимог до програмного забезпечення із використанням певних методів, засобів та існуючих технологій.
8. Здійснити аналітичний огляд існуючих методів формалізації вимог до програмного забезпечення.
9. Удосконалити формалізації вимог до програмного забезпечення із використанням оцінювання якості результатів опитування стейкголдерів.

Практичне значення отриманих результатів полягає в тому, що її результати та теоретичні напрацювання можуть використовуватись у роботі розробників програмного забезпечення під час аналітичного процесу щодо визначення вимог та формулювання технічного завдання.

Результати даної магістерської роботи можуть бути використані у подальших наукових дослідженнях, а також аналітиками у сфері IT-розробки під час проектування програмного забезпечення.

1.5 Висновки до 1-го розділу

Перший розділ містить аналітичну складову дослідження та структуризації предметної області, а саме увагу приділено формалізації вимог до програмного забезпечення. В результаті такого дослідження було структуровано поняття вимог до програмного забезпечення, а також визначено важливість та доцільність стандартизації цих вимог та їх впливу на подальшу розробку програмного забезпечення.

Також здійснено аналіз методів та засобів, що допомагають стандартизувати та формалізувати вимоги до програмного забезпечення впродовж всього життєвого циклу програмного продукту.

Також окреслено мету роботи, що полягає в удосконаленні методу формалізації вимог до програмного забезпечення із використанням оцінювання якості результатів опитування зацікавлених сторін.

2 МЕТОД ФОРМАЛІЗАЦІЇ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ НА ОСНОВІ ОЦІНКИ ЯКОСТІ РЕЗУЛЬТАТІВ ОПИТУВАННЯ ЗАЦІКАВЛЕНИХ СТОРІН

2.1 Модель процесу формалізації вимог до програмного забезпечення через оцінювання якості опитування зацікавлених сторін

Концептуальне моделювання в аспекті формалізації вимог до програмного забезпечення із використанням оцінювання якості опитування – це процес створення абстрактного, незалежного від технологій та реалізації, інтелектуального представлення предметної області та бажаної системи. Воно слугує мостом між неформальними уявленнями та потребами зацікавлених сторін (ЗС) і строгою, формальною специфікацією, необхідною для розробників.

У цьому контексті концептуальне моделювання має такі ключові аспекти, як абстракція знань, уточнення та усунення неоднозначності, інтеграція якості опитування, основа для формалізації.

Абстракція знань полягає у тому, що модель витягує суттєві сутності, атрибути та відносини з неструктурованих відповідей ЗС. Вона структурує хаотичні відомості природної мови в чіткий, логічно організований набір концепцій (наприклад, Користувач, Замовлення, Транзакція).

Процес уточнення та усунення неоднозначності показує, що концептуальна модель є першим кроком до формалізації і допомагає виявити та вирішити суперечності та двозначності, які неминуче виникають у процесі опитування ЗС, особливо враховуючи різну якість цих опитувань.

Інтеграція якості опитування, бо модель є центральним місцем, де результати оцінювання якості опитування використовуються для зважування та пріоритезації різних вимог. Це гарантує, що найважливіші та найнадійніші вимоги, отримані від найбільш компетентних зацікавлених сторін, займають центральне місце в моделі.

Основа для формалізації, бо модель виступає як проміжна ланка, яка перетворює неформальні вимоги на напівформальну структуру, яку потім можна безпосередньо трансформувати у формальну специфікацію, наприклад, за допомогою UML, OCL або LTL для детальної розробки.

Таким чином, концептуальне моделювання є фундаментальним інструментом для осмислення та структурування складної предметної області, забезпечуючи, що кінцеві вимоги до програмного заьезпечення є не лише повними, але й достовірними, завдяки інтеграції показників якості джерел інформації.

Важливою є концептуальна модель методу формалізації вимог до розробки програмного забезпечення, що інтегрує оцінювання якості опитування зацікавлених сторін (ЗС) та зосереджена на зменшенні двозначності та підвищенні точності вимог, отриманих з неформальних джерел та неявних знань зацікавлених сторін. Цей метод розглядає опитування не лише як інструмент збору даних, але і як об'єкт оцінювання якості, який безпосередньо впливає на надійність кінцевої формалізованої специфікації. Загалом така модель включає наступні етапи: виявлення наявних вхідних даних та етап збору вимог (оцінювання якості опитування, якість процесу опитування, якість джерела); етап аналізу та уточнення вимог; етап формалізації вимог; зворотній зв'язок та валідація.

На етапі виявлення наявних вхідних даних та збору вимог відбувається збір вимог через опитування зацікавлених сторін. Тут вхідними даними можуть бути неформальні відповіді зацікавлених сторін, виражені природною мовою, неявні знання, а також додаткові артефакти опитування, наприклад, записи, транскрипції, нотатки).

Оцінювання якості опитування (ОЯО) відбувається одночасно з опитуванням або безпосередньо після нього проводиться оцінювання якості. Оцінювання якості опитування фокусується на двох ключових аспектах, а саме якості процесу опитування, коли оцінюється чіткість запитань, послідовність,

повнота охоплення предметної області, відсутність навідних питань та якості джерела (ЗС), яке полягає у тому, що оцінюється рівень знань, досвід у предметній області, авторитет зацікавлених сторін та її схильність до послідовних відповідей.

2. Етап аналізу та уточнення вимог полягає у тому, що отримані неформальні вимоги та результати оцінювання якості опитування інтегруються для підвищення якості подальшого аналізу.

Для виявлення неоднозначностей та суперечностей відповіді зацікавлених сторін аналізуються на предмет наявності двозначності, нечіткості та суперечностей між вимогами, висловленими різними ЗС.

Вагові коефіцієнти якості (WQC). На основі результатів ОЯО, кожній отриманій вимозі або кожній ЗС призначається ваговий коефіцієнт якості (WQC). Вимоги, отримані з якісно проведеного опитування та від більш компетентних ЗС, отримують вищий WQC.

Вимоги з низьким WQC автоматично позначаються як високопріоритетні для уточнення. ЗС можуть бути опитані повторно, або їхні відповіді піддаються детальнішому аналізу з використанням інструментів обробки природної мови (NLP) для нормалізації термінології.

3. Етап формалізації вимог перетворює уточнені вимоги на формальну специфікацію, використовуючи встановлені вагові коефіцієнти. Застосування формальних методів, коли уточнені вимоги трансформуються у формальну мову моделювання (наприклад, UML/OCL, LTL, або спеціалізовану мову специфікацій). WQC використовується для вирішення конфліктів. Наприклад, якщо дві вимоги відрізняються, але вимога з вищим WQC має пріоритет у прийнятті рішення щодо кінцевої формальної структури.

Генерація формалізованої специфікації є кінцевим результатом формалізованої специфікація вимог, що є однозначною, узгодженою та простежуваною. Формалізованої специфікація вимог містить формальні

визначення, включаючи перед- та постумови, інваріанти та обмеження, що мінімізує інтерпретаційні помилки на етапі розробки.

4. Зворотний зв'язок та валідація.

Після формалізації специфікація вимог має бути валідована для підтвердження її відповідності потребам ЗС.

Валідація ЗС. Хоча ФСР є формальною, вона має бути подана ЗС у зрозумілому для них вигляді, наприклад, через схеми, діаграми або прототипи).

КоригуванняЯкщо валідація виявляє невідповідності, процес повертається на етап збору або уточнення, при цьому WQC може бути переглянуто для подальшого підвищення якості опитування та аналізу.

Ця модель створює петлю якості, де якість вихідних даних (опитування) систематично використовується для підвищення надійності процесу формалізації, забезпечуючи, що фінальні формальні вимоги є не лише точними, але й релевантними та правильними з точки зору потреб найкомпетентніших ЗС.

На рисунку 2.1 подано концептуальну модель для процесу формалізації вимог до програмного забезпечення на основі опитування зацікавлених сторін.

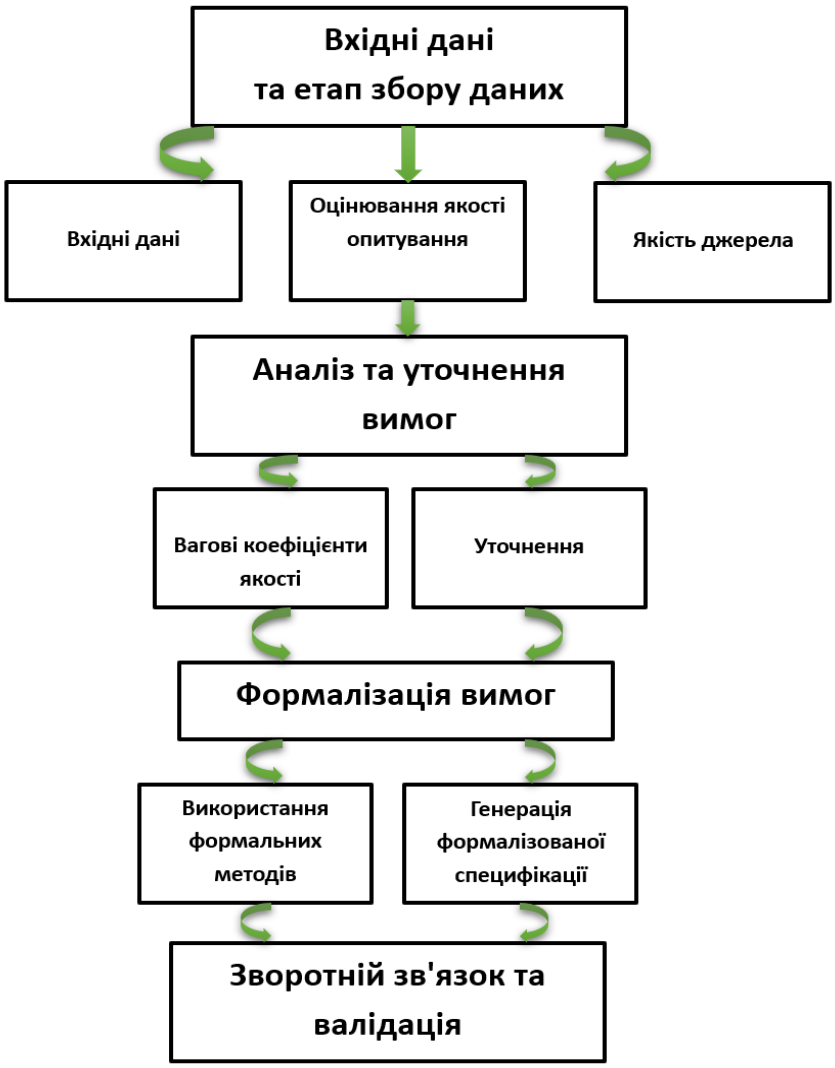


Рисунок 2.1 – Концептуальна модель

2.2 Метод формалізації вимог до програмного забезпечення на основі оцінки якості результатів опитування зацікавлених сторін

Метод формалізації вимог до програмного забезпечення на основі оцінювання якості результатів оцінювання зацікавлених сторін - це ітеративний процес, який фокусується не лише на зборі думок зацікавлених сторін (стейкголдерів), а й на аналізі надійності, узгодженості та ясності їхнього фідбеку перед тим, як перетворювати вимоги у формальну модель.

Головна ідея полягає в тому, щоб формалізувати лише ті вимоги, які пройшли фільтр якості, заснований на фідбеку зацікавлених сторін. Це дозволяє

уникнути проблеми сміття на вході - сміття на виході, коли у формальну модель закладаються нечіткі, суперечливі або неперевірені ідеї.

Цей метод можна розбити на кілька ключових етапів.

1. Створення артефактів для оцінювання, тобто на основі початкових, неформальних вимог (зібраних через інтерв'ю, опитування) створюються проміжні представлення вимог, або артефакти. Це можуть бути, наприклад, прототипи інтерфейсу - візуальні макети, що показують, як виглядатиме система.

Історії користувачів - короткі описи функціональності з точки зору користувача. Діаграми процесів - схеми, що ілюструють бізнес-логіку. Текстові специфікації - більш деталізовані описи функцій.

2. Оцінювання артефактів стейкхолдерами, коли групам стейкхолдерів (користувачі, менеджери, технічні експерти) пропонують оцінити ці артефакти за допомогою структурованих методів, наприклад:

Анкетування, коли стейкхолдери виставляють оцінки, наприклад, від 1 до 5 за різними критеріями: важливість, терміновість, зрозумілість.

Ранжування, коли просять розставити вимоги за пріоритетом. Воркшопи, піж час яких проводиться обговорення, де фіксуються коментарі, зауваження та пропозиції.

3. Аналіз якості результатів оцінювання, що відноситься до ключового етапу, тобто зібраний фідбек не просто приймається на віру, а проходить через аналіз якості за кількома параметрами, що описано нижче.

Консистентність (узгодженість), що поділяється на внутрішню та зовнішню. Внутрішня відповідає на запитання чи не суперечить стейкхолдер сам собі у різних відповідях. Зовнішня відповідає на запитання наскільки узгодженими є думки різних зацікавлених сторін або їх груп. Наприклад, якщо всі розробники кажуть, що вимога Х складна, а всі менеджери - що вона проста, це низька консистентність.

Рівень консенсусу показує який відсоток зацікавлених сторін погоджується з певною вимогою або її пріоритетом. Високий консенсус є індикатором високої якості.

Чіткість та специфічність фідбеку показує наскільки конкретними є коментарі. Наприклад, *Мені не подобається дизайн* - це фідбек низької якості. Кнопка *Купити занадто мала і розташована в неочікуваному місці* - високої.

Вага або авторитет зацікавленої особи, коли думка головного архітектора щодо технічної реалізації має більшу вагу, ніж думка маркетолога. Думка кінцевого користувача про зручність інтерфейсу важливіша за думку системного адміністратора. Кожній зацікавленій особі або групі може бути присвоєно коефіцієнт ваги для кожної конкретної вимоги.

4. Прийняття рішення про формалізацію. На основі аналізу якості кожна вимога отримує умовний рейтинг довіри. Високий рейтинг, тобто вимога має узгоджений, чіткий фідбек від авторитетних стейкхолдерів. Її можна формалізувати, наприклад, описувати мовою UML+OCL, Z або TLA+.

Середній рейтинг, коли є невеликі розбіжності або нечіткості. Вимога відправляється на додаткове обговорення з конкретною групою зацікавлених сторін для уточнення.

Низький рейтинг, коли існують серйозні суперечності, фідбек нечіткий, немає консенсусу. Формалізація блокується. Вимога повертається на самий перший етап для повного переосмислення. Цей цикл повторюється, доки не буде досягнуто достатнього рівня довіри для більшості ключових вимог. Як практичний приклад можна привести проєкт розробки мобільного банкінгу із вимогою мати можливість відкрити депозит через застосунок.

В якості артефакту створено прототип з трьома екранами процесу. Під час оцінювання прототип показують трьом групам: кінцеві користувачі, тобто клієнти, менеджери з продуктів, юристи. У результаті фідбеку виявлено, що користувачі виявили, що процес занадто довгий, багато полів. Чіткість даного формулювання - середня. Менеджери озвучили, що треба додати опцію вибору

капіталізації відсотків. Чіткість даного формулювання висока. За словами юристів формулювання договору на останньому кроці не відповідає законодавству. Показано високу чіткість, та максимальний результат.

5. Аналіз якості, тобто консенсус щодо необхідності функції високий. Консистентність думок щодо процесу низька (користувачі хочуть простіше, менеджери - складніше). Фідбек юристів має найвищий пріоритет.

6. Рішення, коли юридичні аспекти формалізуються негайно як жорсткі бізнес-правила. Питання про капіталізацію формалізується як вимога до бізнес-логіки. Процес заповнення даних повертається на доопрацювання для пошуку компромісу між простотою для користувачів і потребами менеджерів.

Перевагами є:

– зниження ризиків, бо зменшує ймовірність розробки функціоналу, який нікому не потрібен або реалізований неправильно;

– об'єктивність - рішення про пріоритети та зміст вимог приймаються на основі даних, а не інтуїції;

– фокус на важливому, що дозволяє зосередити зусилля на найбільш узгоджених і зрозумілих вимогах.

До недоліків можна віднести:

– часові витрати, тобто процес є довшим і вимагає більше ітерацій;

– складність, яка потребує навичок у фасилітації, аналізі даних та роботі зі стейкхолдерами;

– ризик *втоми* стейкхолдерів, бо постійні опитування та воркшопи можуть знизити залученість зацікавлених сторін.

Загалом метод формалізації вимог до програмного забезпечення з наукової точки зору є гібридною системою, що ґрунтується на комбінації фундаментальних методів з кількох дисциплін, а не єдиним монолітним підходом. Нижче подано низку методів, на основі якого було здійснено вдосконалення розроблюваного методу в даному дослідженні: методи інженерії

програмного забезпечення, математичні основи з теорії рішень, методи комп'ютерної лінгвістики, адаптація через машинне навчання.

По-перше, вся його структура походить з інженерії вимог (Requirements Engineering). ReqQual є інструментом, що автоматизує класичні процеси цієї дисципліни. Він вирішує проблеми виявлення, беручи суб'єктивний метод опитування та об'єктивізуючи його. Модуль Formalizer безпосередньо реалізує методи специфікації, перетворюючи неформальний текст на структуровані артефакти, як-от User Stories. Цикл зворотного зв'язку, де стейкхолдер підтверджує інтерпретацію, є прямим застосуванням валідації вимог (перевірки, «чи ми будуємо правильну річ»). Найголовніше, вимога простежуваності реалізує один з ключових методів програмної інженерії - управління простежуваністю (Requirements Traceability), зв'язуючи ідею з кодом.

Математичне ядро фреймворку, зокрема Q-Score Engine, базується на теорії прийняття рішень та багатокритеріальному аналізі рішень (MCDA). Сама формула Q-Score є класичною реалізацією методу зваженої суми (Weighted Sum Model). Цей метод дозволяє брати різні, незрівнянні критерії, наприклад авторитет та чіткість, нормалізувати їх до єдиної шкали та застосовувати ваги для отримання інтегральної оцінки. Водночас, Priority-Score (RICE/ICE) є прямим застосуванням мікроекономічного аналізу витрати-вигоди (Cost-Benefit Analysis), де Effort - це витрати, а Impact та Reach - очікувана вигода.

Здатність розуміти вхідні дані забезпечується методами обробки природної мови (NLP). Модуль Ingest використовує розпізнавання іменованих сутностей (NER) для вилучення ключових об'єктів, наприклад користувач, API. Розрахунок *W_Clarity* ґрунтується на лексичному аналізі для пошуку слів-паразитів. Найскладніший компонент, *W_Consistency*, спирається на дистрибутивну семантику. Цей метод, що використовує векторні представлення слів (Word Embeddings), базується на гіпотезі, що слова зі схожим значенням зустрічаються у схожих контекстах. Розрахунок семантичної схожості - це математичне порівняння цих векторів у багатовимірному просторі.

Нарешті, здатність фреймворку до еволюції походить від методів машинного навчання. Адаптивна модель реалізує навчання з вчителем (Supervised Learning). Пост-релізні дані (кількість багів, успіх фічі) діють як мітки (labels), а система використовує методи регресії або класифікації для коригування ваг Q-Score, щоб краще передбачати успішність вимоги. Навіть Formalizer у своїй основі використовує простішу AI-техніку - експертну систему на основі правил (Rule-Based System), яка застосовує шаблони (якщо... тоді...) для генерації артефактів.

Метод формалізації вимог до програмного забезпечення із використанням оцінювання якості результатів опитування зацікавлених осіб, вдосконалення якого полягає у введенні та визначенні із подальшим опрацюванням неявних вимог зацікавлених сторін по суті, полягає у тому, що треба автоматизувати перетворення хаосу (хочу, щоб було гарно) на чітке технічне завдання (GIVEN/WHEN/THEN), при цьому відфільтрувавши 90% так званого шуму від зацікавлених сторін (стейкхолдерів).

Даний метод грунтується на основі багатокритеріальний аналізу рішень (MCDA). MCDA - це методологія, призначена для підтримки прийняття рішень, де вибір найкращої альтернативи залежить від оцінки за кількома, часто конфліктуючими, критеріями.

З математичної точки зору, метод багатокритеріального аналізу рішень (MCDA) у фреймворку ReqQual найкраще описується через зважену суму або функцію корисності (utility function). Цей підхід дозволяє перетворити вектор індивідуальних оцінок вимоги на єдиний скалярний показник *Q-Score*.

Загальна модель (метод зваженої суми) полягає у тому, що основою для розрахунку Q-Score є метод зваженої суми (Weighted Sum Model - WSM). Кожна вимога R_i оцінюється за n критеріями C_j, де кожен критерій має присвоєну вагову коефіцієнт v_j.

Фінальний (індекс якості) *Q-Score* для вимоги обчислюється як:

$$Q - Score(R_i) = \sum_{j=1}^{n}(v_j \cdot N_j(R_i)) \quad (1)$$

де:

$Q - Score(R_i)$ - фінальний інтегральний показник якості для вимоги;

n - загальна кількість критеріїв оцінки (у нашому випадку:

v_j - ваговий коефіцієнт (weight) для критерію. Ці коефіцієнти визначають відносну важливість кожного критерію, і їх сума повинна дорівнювати одиниці;

$N_j(R_i)$ - нормалізоване значення показника вимоги (R_i) за критерієм j. Це значення $\in= [0,1]$, де 1 означає ідеальну якість.

Специфічні критерії та розрахунки полягають у тому, що у фреймворку ReqQual ключові критерії $N_j(R_i)$ генеруються з різноманітних джерел, але завжди нормалізуються до інтервалу $[0,1]$, і передбачають нормалізовану оцінку авторитету, нормалізовану оцінку чіткості, нормалізовану оцінку узгодженості.

Нормалізована оцінка авторитету N_{Auth} визначається на основі ролі зацікавленої сторони S_k. Це дискретне або категорійне значення, яке попередньо задано в і прямо нормалізовано, як це показано у (2).

$$N_{Auth}(R_i) = Matrix(S_k) \in [0,1] \quad (2)$$

Нормалізована оцінка чіткості $N_{Clarity}$ є критерієм, що виступає функцією лінгвістичного аналізу, що використовує Computational Linguistics для виявлення специфічності. Його можна змоделювати як це подано у (3):

$$N_{Clarity}(R_i) = \max(0.1 - \sum_m \alpha_m \cdot I_m + \sum_k \beta_k \cdot j_k) \quad (3)$$

де:

I_m - бінарний індикатор, що дорівнює 1, якщо слово-паразит m присутнє (наприклад, зручно, швидко);

α_m - штрафний коефіцієнт за слово m;

j_k - бінарний індикатор, що дорівнює 1, якщо присутній кількісний показник (наприклад, 100 мс, 5 користувачів);

β_k - бонусний коефіцієнт за показник k.

Нормалізована оцінка узгодженості N_{cons}, де критерій Consistency є функцією семантичної близькості, що базується на дистрибутивній семантиці (векторних моделях слів, embeddings), як це показано у (4).

$$N_{cons}(R_i) = f(Similarity\ R_i, D)) \quad (4)$$

де:

D - множина усіх існуючих формалізованих вимог;

$(Similarity\ R_i, D))$ - це функція, яка розраховує семантичну схожість R_i з D наприклад, використовуючи косинусну схожість між векторними представленнями вимог;

f(x) - функція трансформації. Наприклад, якщо схожість занадто висока (приблизно рівна 1), то N_{cons} може отримати штраф (потенційний дублікат), а якщо виявлено семантичний конфлікт (спеціалізованим алгоритмом), N_{cons} буде низьким (приблизно дорівнюватиме 0).

Таким чином, MCDA забезпечує математично обґрунтований механізм для ранжування та фільтрації вхідних вимог на основі їхньої внутрішньої якості, дозволяючи системі автоматично приймати рішення про подальшу формалізацію.

2.3 Алгоритм формалізації вимог до програмного забезпечення на основі оцінки якості результатів опитування зацікавлених сторін

Для впровадження розробленого методу формалізації вимог до програмного забезпечення доцільно показати алгоритм, за яким можлива реалізація вдосконаленого методу підтримки.

Першим кроком є алгоритм роботи фреймворку, що виступає послідовним конвеєром, який інтегрує лінгвістичний аналіз, багатокритеріальне оцінювання та трансформацію даних. Його мета - автоматично фільтрувати низькоякісні вимоги перед тим, як вони потраплять до команди розробки.

Алгоритм роботи фреймворку складається із декількох кроків. Нижче наведено покроковий опис процесу, який охоплює всі модулі.

Крок 1 - інджестування та лінгвістична підготовка, коли система приймає сиру вимогу та ідентифікатор зацікавленої сторони. Застосовується обробка природної мови для нормалізації тексту. Далі генеруються лінгвістичні метрики: витягуються сутності, підраховується кількість маркерів неоднозначності та кількісних показників. Також генерується векторне представлення вимоги для семантичного порівняння.

Крок 2 - обчислення Q-Score (MCDA-Core) де відбувається розрахунок субоцінок, який складається із таких етапів:

- витягується нормалізоване значення авторитету;
- розраховується на основі штрафів за неоднозначність та бонусів за кількісні показники;
- обчислюється семантична схожість з базою існуючих вимог для виявлення потенційних конфліктів або дублювання.

Агрегація (WSM) відбувається шляхом об'єднання субоцінки за методом зваженої суми для отримання фінального значення.

Крок 3 - прийняття рішення та фільтрація, де відбувається порівняння з порогом, тобто порівнюється із заздалегідь визначеним порогом якості.

Вводиться так звана гілка відхилення (Якщо..то), то тоді вимога автоматично відхиляється. Система генерує зворотний зв'язок на основі критерію, який отримав найнижчий бал, наприклад, *Низький: вкажіть*

конкретні часові показники. Процес завершується, вимога повертається на доопрацювання. Також є гілка продовження (Якщо), то тоді вимога переходить до наступного етапу.

На четвертому кроці відбувається пріоритезація та формалізація. Пріоритезація (RICE) полягає у тому, що уповноважений аналітик або техлід вводить суб'єктивні оцінки для RICE/ICE (Reach, Impact, Confidence, Effort). Розраховується .

Трансформація (MDE/Правила) відбувається тоді, коли система застосовує правила формалізації до високоякісного тексту, щоб перетворити його на стандартизований артефакт, наприклад, User Story: Як користувач... Я хочу... Щоб...). Валідація полягає у тому, що система надсилає формалізований артефакт на підтвердження.

На п’ятому кроці відбувається реєстрація та зворотний зв'язок (Registry & Feedback Loop). Реєстрація відбуважться, коли фінальний артефакт зберігається в реєстрі. Таким чином створюється наскрізний зв'язок з оригінальним ID, ID в системі управління проєктами (наприклад, Jira) та ID коміту в репозиторії.

Адаптація (ML Feedback Loop) полягає у тому, що після випуску фічі система збирає фактичні метрики успішності, наприклад, кількість багів, рівень використання. Ці дані використовуються для перенавчання моделі та динамічного коригування вагових коефіцієнтів, забезпечуючи постійне підвищення точності оцінки.

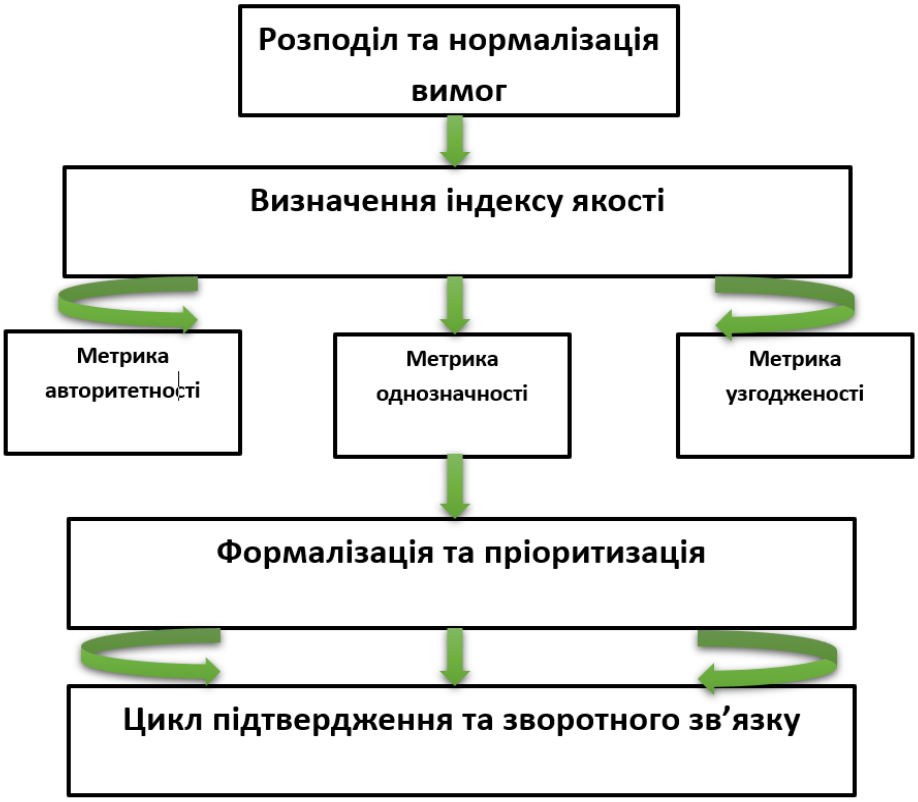


Рисунок 2.3 – Алгоритм формалізації вимог до програмного забезпечення

Тобто алгоритм роботи даного фреймворку грунтується на використанні запропонованого методу та має декілька ключових етапів, а саме етап інжестування та лінгвістична декомпозиція

На етапі інжестування та лінгвістичної декомпозиції відбувається не просто прийом тексту, а його декомпозиція на кількісні ознаки (фічі) для подальшої оцінки. Потрібно пройти кроки прийому та асоціації, тобто вимога асоціюється з ідентифікатором стейкхолдера та метаданими, наприклад, канал подання, час.

Генерація векторного представлення працює із використанням методів дистрибутивної семантики, наприклад, моделі BERT, Word2Vec або їхні контекстні аналоги для перетворення тексту вимоги на векторне представлення у багатовимірному просторі. Це дає змогу математично порівнювати значення вимоги з базою знань.

Лінгвістична фільтрація полягає у тому, що застосовується лексичний аналіз із використанням Part-of-Speech (POS) tagging для більш точного

визначення. Наприклад, виявляється, чи є швидко прислівником, що модифікує дієслово (нечіткість), чи це частина виразу швидкість *200 мілісекунд*, тобто кількісна специфікація, що є критичним для генерації.

На етапі багатокритеріального оцінювання (MCDA Core) виконується метод зваженої суми (WSM). Критерій авторитету працює тоді, коли значення береться з матриці цінності зацікавлених сторін (Stakeholder Value Matrix), яка є попередньо нормалізованою оцінкою, наприклад, для стажера, для архітектора.

Критерій чіткості обчислюється як інверсійна функція від штрафів за неоднозначність та пряма функція від бонусів за кількісні метрики. Всі штрафи та бонуси математично масштабуються, щоб забезпечити, що .

Критерій узгодженості полягає у тому, що система порівнює векторне представлення з векторами всіх вимог у базі.

Використовується метрика косинусної схожості (Cosine Similarity) і отримується низький бал, якщо виявляє:

а) дуже високу схожість або потенційний дублікат;

б) семантичний конфлікт, наприклад, система має бути доступна 24/7 або система має бути вимкнена щодня о 03.00).

Агрегація відбувається, коли фінальні значення приймаються і генерується WSM, де сума вагових коефіцієнтів дорівнює 1 і є нормалізованою, що запобігає домінуванню однієї метрики над іншими.

Ключовою точкою прийняття рішення є етап фільтрації та зворотного в'язку, коли відбувається ухвалення чи відхилення. Тут рішення виступає в ролі первинного фільтра якості.

Динамічна генерація зворотного зв'язку відбувається тоді, коли вимога відхилена і система не просто видає повідомлення про помилку. Вона ідентифікує критерій із найменшим нормалізованим значенням, що дозволяє згенерувати конкретне, дієве повідомлення, наприклад, низький бал . Будь ласка, замініть швидко на кількісний показник. Це перетворює відхилення на цикл навчання для стейкхолдера.

На етапі пріоритезації та формалізації відбувається двоетапна пріоритезація. Фреймворк застосовує двоступеневий фільтр: спочатку фільтр якості, а потім фільтр цінності. Це гарантує, що дорогий людський ресурс, наприклад, час аналітиків та техлідів, витрачається лише на якісні вимоги.

Формалізація відбувається тоді, коли вимога, що пройшла фільтр, трансформується. Система використовує експертні правила для перетворення семантичних маркерів на синтаксично коректний формалізований артефак, наприклад, синтаксис Gherkin або шаблон User Story.

На етапі реєстрації та адаптації (Adaptive Learning Loop) працює наскрізна простежуваність, тобто усі ID створюють карту зв'язків (Unique Identifier Map), що гарантує простежуваність від до ID в системі управління проєктами та до фінального у системі контролю версій.

Також є важливим адаптивне навчання (ML Feedback), коли відбувається перехід від статичного WSM до динамічної моделі. Система використовує навчання з учителем, де:

- ознаки (Features) - це нормалізовані оцінки;
- мітка (Label) - це запізніла метрика успіху, наприклад, , що асоціюється з вимогою після її випуску.

Модель регресія або класифікація навчається передбачати успіх на основі початкових даних. Результат цього навчання використовується для динамічного калібрування вагових коефіцієнтів у формулі, постійно покращуючи точність прогнозування якості вимог.

2.4 Висновки до другого розділу

Отже, другий розділ містить результати концептуального моделювання, в результаті якого було отримано концептуальну модель. Запропонована

концептуальна модель процесу формалізації вимог до програмного забезпечення відходить від традиційного лінійного підходу і ставить у центр уваги критичне оцінювання якості комунікації із зацікавленими сторонами. В основі цієї моделі лежить гіпотеза, що якість кінцевих формалізованих специфікацій не може перевищувати якість так званої сирої інформації, отриманої під час усних чи письмових опитувань, тому процес формалізації має починатися не після завершення інтерв'ю, а інтегруватися безпосередньо в процес його оцінювання.

Початковий етап моделі невизначений і носіями потреб є зацікавлені сторони, чиє розуміння проблем часто буває неструктурованим, емоційним або суперечливим через складний бізнес-контекст. Активна фаза моделі розпочинається з процесу безпосереднього проведення інтерв'ю, воркшопів та анкетування, результатом чого стають так звані сирі дані опитування у вигляді нотаток, аудіозаписів та транскрипцій, які ще не є готовими вимогами.

Центральним елементом і своєрідним рушієм усієї моделі виступає фільтр якості отриманої інформації. Перш ніж перейти до будь-якої формалізації, сирі дані піддаються ретельному мета-аналізу, де оцінюється не стільки зміст бізнес-потреби, скільки якість проведеного опитування. Аналітик перевіряє отримані відповіді на відповідність низці критичних критеріїв, визначаючи, чи була забезпечена повнота охоплення предметної області і чи не залишилося білих плям. Критично важливою є перевірка на однозначність, адже відповіді, що допускають множинні інтерпретації, вважаються неякісною сировиною. Також дані аналізуються на предмет внутрішньої несуперечливості, релевантності цілям проєкту та загального рівня довіри до джерела інформації.

У цей момент настає ключова точка прийняття рішення, яка визначає подальший рух процесу. Якщо сирі дані не проходять фільтр якості через неоднозначність або неповноту, процес формалізації блокується, і активується перша петля зворотного зв'язку, що повертає аналітика до етапу опитування для проведення уточнюючих, більш сфокусованих інтерв'ю. Лише після того, як

інформація досягає необхідного рівня якості та однозначності, вона допускається до етапу трансформації.

На етапі формалізації якісні, перевірені сирі дані перекладаються з природної мови на інженерну, перетворюючись на моделі процесів, структуровані історії користувачів, варіанти використання та чітко визначені атрибути якості, формуючи чернетку специфікацій. Завершальний цикл моделі передбачає валідацію та верифікацію цих чернеток зацікавленими сторонами, щоб підтвердити правильність інтерпретації їхніх слів. У разі виявлення помилок інтерпретації спрацьовує друга, коротша петля зворотного зв'язку для корекції моделей, і лише після успішного підтвердження формується фінальна базова лінія вимог, готова до передачі в розробку. Таким чином, ця модель змушує аналітика діяти як діагност, постійно оцінюючи не тільки те, що говорять клієнти, а й те, наскільки якісно вони це роблять, що дозволяє виявляти ризики на найбільш ранніх стадіях проєкту.

Також було описано удосконалений метод, заснований на використанні багатокритеріального аналізу рішень (Multi-Criteria Decision Analysis - MCDA), застосований у контексті інженерії вимог (Requirements Engineering).

Основним науковим методом, на якому ґрунтується формалізація вимог через оцінювання якості результатів опитування є багатокритеріальний аналіз рішень - це методологія, призначена для підтримки прийняття рішень, де вибір найкращої альтернативи залежить від оцінки за кількома, часто конфліктуючими, критеріями. У контексті ReqQual це реалізується через Q-Score Engine, який перетворює суб'єктивні та якісні вхідні дані, тобто результати опитування на кількісний, агрегований показник.

Саме розрахунок Q-Score є прямою реалізацією MCDA, найчастіше у формі методу зваженої суми (Weighted Sum Model - WSM). Цей метод дозволяє системі одночасно оцінювати вимогу за незрівнянними критеріями, такими як авторитет джерела *W_Authority*, що є метрикою управління зацікавленими сторонами, та чіткість висловлювання *W_Clarity,* що є лінгвістичною метрикою.

Шляхом нормалізації та присвоєння ваг, MCDA дає змогу визначити, якій вимозі слід приділити ресурси для подальшої, дорогої, формалізації.

Для функціонування MCDA використовуються інші науково-технічні методи як джерела даних, а саме обробка природної мови (NLP). Саме методи обчислювальної лінгвістики надають MCDA необхідні метрики. Методи лексичного аналізу використовуються для генерації метрики *W_Clarity* шляхом виявлення неоднозначних термінів. Водночас, дистрибутивна семантика через векторні представлення слів використовується для обчислення *W_Consistency,* оцінюючи семантичну схожість або конфлікт із вже існуючою базою знань.

Модель-керована інженерія (Model-Driven Engineering - MDE). Цей підхід є основою модуля формалізації. Після того, як MCDA підтвердив високу якість вхідних даних, MDE-подібний рушій правил бере цю якісну модель, тобто вимогу з високим Q-Score і трансформує її в інший, більш структурований формат, наприклад, User Story або Gherkin, що є придатним для розробки.

Отже, MCDA визначає, що потрібно формалізувати, а NLP надає кількісні дані, необхідні для цього рішення.

3 АРХІТЕКТУРА СИСТЕМИ ФОРМАЛІЗАЦІЇ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1Формування та аналіз вимог програмної реалізації системи формалізації вимог до програмного забезпечення

Загальні вимоги до програмного забезпечення, призначеного для автоматизації процесу формалізації вимог на основі аналізу даних опитувань зацікавлених сторін, мають охоплювати весь життєвий цикл обробки інформації від її отримання до генерації фінальних специфікацій, приділяючи особливу увагу інтелектуальному аналізу якості вхідних даних.

На етапі збору та попередньої обробки даних система повинна забезпечувати мультимодальне введення інформації, підтримуючи імпорт структурованих відповідей з анкет, неструктурованих текстових транскриптів інтерв'ю, а також бажано аудіозаписів із інтегрованою функцією розпізнавання мовлення. Критично важливою є здатність системи виконувати попередню нормалізацію цих даних, включаючи очищення від шуму, ідентифікацію спікерів та сегментацію діалогів на логічні блоки, що стосуються конкретних бізнес-процесів чи функцій.

Ядро функціональних вимог має зосереджуватися на інтелектуальному аналізі якості комунікації, що вимагає використання передових методів обробки природної мови та машинного навчання для виходу за межі простого сумування тексту. Система повинна автоматично здійснювати діагностику отриманих відповідей, виявляючи та позначаючи маркери низької якості, такі як семантична неоднозначність, використання нечітких термінів, внутрішні суперечності у твердженнях одного або кількох респондентів, а також неповноту інформації стосовно визначених атрибутів необхідної функціональності. Результатом цього етапу має бути не просто набір сирих даних, а розмічений корпус інформації з рейтингом довіри до кожного

твердження та сформованим переліком питань для уточнення, що ініціює петлі зворотного зв'язку з аналітиком.

Вимоги до блоку трансформації та генерації передбачають здатність системи перетворювати верифіковані, якісні фрагменти природної мови у стандартизовані артефакти вимог. Програмне забезпечення повинно вміти автоматично генерувати чернетки User Stories із відповідними критеріями приймання, структуровані сценарії Use Cases із виділенням основних та альтернативних потоків, а також потенційно створювати візуальні моделі, наприклад, діаграми діяльності UML або BPMN) на основі текстових описів процесів, забезпечуючи при цьому наскрізне трасування від фінальної вимоги до вихідної цитати стейкхолдера.

Для забезпечення ефективної взаємодії людини та машини система повинна надавати аналітику ергономічний інтерфейс для керування робочим процесом. Це включає інструменти для рецензування результатів автоматичного аналізу якості, можливість підтверджувати або відхиляти знайдені системою неоднозначності, а також потужний редактор для пост-обробки згенерованих чернеток вимог, який дозволяє швидко коригувати тексти та моделі перед їх фінальним затвердженням, зберігаючи історію всіх змін.

Нарешті, нефункціональні вимоги повинні гарантувати високий рівень безпеки та конфіденційності даних, оскільки інформація, отримана під час інтерв'ю, часто є чутливою комерційною таємницею, що вимагає шифрування та суворого контролю доступу. Крім того, система має бути інтегрованою в існуючу екосистему розробки, забезпечуючи можливість експорту затверджених вимог через API або прямі конектори до популярних систем управління проєктами та вимогами, таких як Jira, Azure DevOps або Confluence.

На основі попередніх міркувань та розробки прототипу, можна сформулювати структурований перелік вимог до фреймворку ReqQual.

Функціональні вимоги (FR) для кожного модуля включають точний перелік.

для модуля інджестування (Ingest):

– система повинна надавати інтерфейс (API та/або UI) для прийому сирих вимог у вигляді тексту.

– кожен запит повинен бути асоційований з ідентифікатором зацікавленої сторони (stakeholder ID);

– забезпечення мультимодальності, тобто система повинна підтримувати прийом супутніх артефактів, пов'язаних з вимогою, зокрема URL-посилання на зовнішні ресурси. наприклад, Figma, Miro; вкладені файли, напрмклад, .png, .pdf, .docx.

– система повинна проводити попередню обробку (pre-processing) тексту для нормалізації.

Для модуля аналізу та скорингу (Q-Score Engine) система повинна використовувати NLP-моделі для аналізу тексту вимоги, зокрема для:

– визначення ключових сутностей;

– лексичного аналізу для виявлення слів-паразитів, наприклад, швидко, зручно;

– виявлення кількісних показників, наприклад, 200 мс, 5 користувачів.

В модулі Q-Score система повинна автоматично розраховувати інтегральний Q-Score (індекс якості) для кожної вимоги, базуючись щонайменше на трьох субоцінках:

– W_Authority (вага авторитету) - оцінка на основі ролі зацікавленої сторони;

– W_Clarity (вага чіткості) - оцінка на основі відсутності слів-паразитів та наявності кількісних показників.

– W_Consistency (вага узгодженості) - оцінка на основі семантичного порівняння з існуючою базою формалізованих вимог (виявлення конфліктів або дублікатів).

Важливим моментом є наявність пріоритезація, тобто система повинна надавати інтерфейс для розрахунку Priority-Score, наприклад, RICE/ICE для вимог, що пройшли фільтр Q-Score.

Інтерфейс повинен дозволяти уповноваженим користувачам, наприклад, аналітикам, техлідам вводити показники Impact, Confidence та Effort.

Модуль формалізації (Formalizer) повинен забезпечувати:

– фільтрацію, тобто система повинна автоматично відхиляти або відправляти на доопрацювання вимоги, Q-Score яких нижчий за визначений поріг;

– зворотний зв'язок, тобто при відхиленні, система повинна інформувати ініціатора про причину, наприклад, низька чіткість;

– генерацію, тобто для вимог, що пройшли фільтр, система повинна генерувати формалізований артефакт, наприклад, User Story, Gherkin-сценарій на основі шаблонів;

– валідацію, тобто система повинна підтримувати цикл підтвердження (approval loop), де ініціатор підтверджує, що формалізований артефакт коректно відображає всі дані.

Модуль реєстру та простежуваності (Registry):

– система повинна присвоювати унікальний ID кожній сирій вимозі;

– система повинна підтримувати наскрізний зв'язок (traceability) між сирою вимогою, її Q-Score, формалізованим артефактом, наприклад, ID в Jira та комітами в системі контролю версій.

– система повинна зберігати історію всіх змін та оцінок для аудиту.

Адаптивна модель (ML Feedback Loop):

– система повинна надавати API для прийому пост-релізних даних, наприклад, кількість багів, пов'язаних з вимогою; метрики успішності;

– система повинна використовувати ці дані для періодичного перенавчання моделі Q-Score, коригуючи ваги, наприклад, W_Authority, W_Clarity.

Нефункціональні вимоги мають забезпечувати:

– продуктивність, тобто інджестування сирої вимоги (API call) повинно займати < 500 мс (P95);

– повний цикл розрахунку Q-Score (включно з NLP) повинен виконуватися асинхронно і займати < 15 секунд на одну вимогу;

– конфігурованість, коли адміністратор системи повинен мати можливість конфігурувати: порогове значення Q-Score; матрицю Stakeholder_Matrix (ваги авторитету); словник слів-паразитів (ambiguity words).

– масштабованість, тобто архітектура повинна бути горизонтально масштабованою, наприклад, через мікросервіси або serverless-функції для обробки пікових навантажень, наприклад, під час квартального планування;

– шифрування, тобто усі дані про вимоги, які можуть містити комерційну таємницю повинні бути зашифровані;

– безпеку, тобто система повинна підтримувати контроль доступу на основі ролей (RBAC). Наприклад, тільки аналітики та техліди можуть вводити RICE-оцінки, тільки адміністратори можуть змінювати матрицю авторитету.

– юзабіліті, тобто інтерфейс для подання вимог повинен бути максимально простим. Інтерфейс та RICE-оцінювання повинен бути оптимізований для швидкої роботи аналітиків.

Не менш важливими є також вимоги до інтеграції:

– система повинна надавати RESTful API для всіх основних операцій, наприклад, подання, отримання статусу, RICE-оцінювання;

– система повинна мати вбудовані інтеграції або механізм веб-хуків для двостороннього зв'язку з популярними системами управління проєктами, наприклад, Jira, Azure DevOps, Asana;

– система повинна інтегруватися з корпоративним провайдером ідентичності, наприклад, Azure AD, Okta для автентифікації та отримання ролей користувачів;

– система повинна мати можливість підключення до репозиторіїв, наприклад, GitHub, GitLab.

Отже, як видно функціональність фреймворку ReqQual полягає у функціонуванні чотирьох основних модулів та адаптивного ядра.

Модуль інджестування повинен забезпечувати прийом сирих текстових вимог через API або UI, обов'язково асоціюючи кожен запит з ідентифікатором зацікавленої сторони. Критично важливою є мультимодальна підтримка, що дозволяє системі приймати та пов'язувати з вимогою супутні артефакти, такі як URL-посилання та вкладені файли Завершується цей етап попередньою обробкою тексту для його нормалізації.

Далі, модуль аналізу та скорингу (Q-Score Engine) використовує NLP-моделі для глибокого аналізу тексту. Він повинен витягувати ключові сутності, ідентифікувати слова-паразити, що знижують чіткість та виявляти кількісні показники, що підвищують її. На основі цього аналізу, система автоматично розраховує Q-Score (індекс якості), який є зваженою сумою трьох метрик: W_Authority, W_Clarity та W_Consistency. Окрім Q-Score, цей модуль повинен надавати інтерфейс для розрахунку Priority-Score (наприклад, RICE), дозволяючи користувачам вводити показники Impact, Confidence та Effort.

Модуль формалізації діє як фільтр та генератор та автоматично відхиляє вимоги з Q-Score нижче конфігурованого порогу, надаючи ініціатору зворотний зв'язок про причини, наприклад, низька чіткість. Вимоги, що пройшли фільтр, трансформуються у формалізовані артефакти (User Stories, Gherkin) за допомогою шаблонів. Для забезпечення коректності інтерпретації, система підтримує цикл підтвердження (approval loop) з ініціатором.

Модуль реєстру та простежуваності є інфраструктурною основою. Він присвоює унікальні ID сирим вимогам і забезпечує наскрізну простежуваність від початкового запиту до Q-Score, ID в Jira і, врешті, до комітів у системі контролю версій. Уся історія змін та оцінок повинна зберігатися для аудиту.

Нарешті, адаптивна модель (ML Feedback Loop) забезпечує еволюцію системи. Фреймворк повинен мати API для прийому пост-релізних даних (наприклад, кількість багів) і використовувати їх для періодичного перенавчання моделі Q-Score, адаптуючи її ваги до реальних результатів.

Нефункціональні аспекти є критичними для життєздатності системи. Продуктивність є ключовою, бо операція прийому запиту повинна виконуватись менш ніж за 500 мс (P95), тоді як повний асинхронний розрахунок Q-Score з NLP не повинен перевищувати 15 секунд.

Система повинна бути високо конфігурованою, дозволяючи адміністраторам налаштовувати поріг Q-Score, матрицю авторитету (Stakeholder_Matrix) та словники слів-паразитів.

Масштабованість має бути забезпечена через горизонтально масштабовану архітектуру, наприклад, мікросервіси, готову до пікових навантажень.

Безпека вимагає шифрування всіх даних та впровадження контролю доступу на основі ролей, щоб, наприклад, RICE-оцінки могли вводити лише аналітики та техліди. Юзабіліті передбачає максимальну простоту інтерфейсу.

Фреймворк не може існувати у вакуумі. Він повинен надавати RESTful API для всіх операцій. Необхідні вбудовані інтеграції для двостороннього зв'язку з системами управління проєктами. Система повинна інтегруватися з корпоративними провайдерами ідентичності для автентифікації та отримання ролей. Також потрібна інтеграція з репозиторіями для реалізації наскрізної простежуваності до комітів.

3.2 Розробка архітектури системи формалізації вимог до програмного забезпечення

Для того, щоб система функціонувала необхідно здійснити проектування її архітектури. Запропонований фреймворк ReqQual (Requirement Quality Framework) є багатокомпонентною програмною системою, призначеною для вирішення фундаментальної проблеми суб'єктивності, неоднозначності та невідповідності при зборі та обробці вимог до програмного забезпечення. Метою системи є трансформація гетерогенних, неструктурованих вхідних даних, в тому числі у формі неявних знань від зацікавлених сторін у формалізовані, пріоритезовані та відстежувані артефакти розробки.

Архітектурно фреймворк являє собою послідовний конвеєр (pipeline) з чотирьох основних етапів, доповнений розширеннями для забезпечення адаптивності та простежуваності (рисунок 3.1).

Перший етап інджестування та нормалізація (Ingest) служить для агрегації вхідних даних з різноманітних джерел (опитувальники, текстові документи, API) та включає парсинг, коли система приймає гетерогенні формати та проводить їх синтаксичний розбір, а також обробку природної мови, тобто є ядром цього етапу. До сирого тексту застосовуються моделі природної мови для:

– вилучення сутностей (Entity Extraction), коли відбувається ідентифікація ключових об'єктів, наприклад, користувач, звіт, кнопка;

– вилучення атрибутів та відносин, коли відбувається ідентифікація бажаних властивостей, наприклад, швидкий, безпечний та зв'язків між сутностями;

– ідентифікація джерела, тобто відбувається асоціація вимоги з конкретною зацікавленою стороною або групою.

Далі відбувається нормалізація, коли усі витягнуті дані перетворюються на стандартизований внутрішній об'єкт RawRequirement, що містить вихідний текст, вилучені сутності та метадані джерела.

На другому етапі відбувається квантифікація якості (Q-Score Engine), що є основним ядром фреймворку, що реалізує власне сам метод оцінки якості.

Замість бінарної оцінки прийнято чи відхилено, він обчислює інтегральний показник Q-Score (Індекс Якості) як зважену суму декількох метрик:

– метрика авторитетності (W_Authority), тобто коефіцієнт, що присвоюється на основі попередньо визначеної матриці ролей. Вимоги від сторін з вищою релевантною експертизою отримують вищу вагу.

– метрика однозначності (W_Clarity), яка розраховується лексико-статистичним аналізом. Наявність кількісних показників, наприклад, менше 200 мс та чітких дієслів підвищує метрику. Наявність слів-амбігнітетів, наприклад, зручний, сучасний застосовує штраф.

– метрика узгодженості (W_Consistency) обчислюється шляхом семантичного порівняння, наприклад, з використанням векторних уявлень тексту нової вимоги з існуючою базою формалізованих вимог. Виявлення прямих протиріч призводить до негативного значення метрики.

– фінальний Q-Score визначає об'єктивну якість формулювання, а не його *важливість*.

На третьому етапі відбувається формалізація та пріоритезація. Даний етап використовує Q-Score як пороговий фільтр.

Під час фільтрація вимоги з Q-Score нижче встановленого порогу автоматично повертаються ініціатору для доопрацювання, формуючи цикл негайного зворотного зв'язку.

На економічному скорингу вимоги, що пройшли Q-Score фільтр, підлягають вторинному скорингу для визначення пріоритету. Фреймворк інтегрує моделі (наприклад, RICE - Reach, Impact, Confidence, Effort), де Impact та Confidence надаються бізнес-аналітиками, а Effort (трудомісткість) – технічною командою. Це дозволяє відокремити *якість* (Q-Score) від *бізнес-цінності* (PriorityScore).

Далі відбувається трансформація, тобто високопріоритетні та якісні вимоги пропускаються через рушій правил (Rules Engine), який на основі класифікації (наприклад, UI-зміна, API, бізнес-логіка) генерує формалізований

артефакт (User Story, Gherkin-сценарій, специфікація OpenAPI) для завантаження в систему управління проєктами.

На четвертому етапі проходить цикл підтвердження та зворотного зв'язку. Формалізований артефакт з третього етапу не одразу надходить у розробку, а повертається ініціатору для фінального затвердження (*Ми інтерпретували Ваш запит таким чином. Підтверджуєте?*). Це замикає короткий цикл, гарантуючи, що інтерпретація системою відповідає початковому задуму.

Для перетворення фреймворку на комплексну платформу, інтегруються наступні розширення:

Розширення 1: Наскрізна простежуваність (End-to-End Traceability), коли система забезпечує нерозривний зв'язок життєвого циклу. Кожному RawRequirement присвоюється унікальний ідентифікатор (req_id), який автоматично інжектується у формалізований тікет (наприклад, Jira) і, через інтеграцію з CI/CD, асоціюється з відповідними комітами в системі контролю версій. Це забезпечує повний аудит та можливість миттєвого відстеження будь-якого рядка коду до його вихідної бізнес-вимоги.

Розширення 2: Адаптивна модель скорингу (ML-Driven Feedback), коли статичні ваги в Q-Score Engine з другого етапу замінюються предиктивною моделлю. Ця модель безперервно навчається, використовуючи пост-релізні дані як навчальну вибірку. Наприклад, якщо вимога з високим Q-Score спричинила велику кількість багів, це використовується як негативний приклад для коригування моделі. Таким чином, система еволюціонує від евристичної до предиктивної, покращуючи точність оцінки якості вимог з часом.

Розширення 3: Мультимодальний агрегатор контексту Модуль Ingest з першого етапу розширюється для обробки мультимодальних даних. Він здатний не лише аналізувати текст, але й ідентифікувати та індексувати пов'язані артефакти, а саме посилання на UI/UX макети (Figma, Miro), вкладені діаграми (UML, BPMN) або таблиці. Наявність таких артефактів автоматично підвищує метрику W_Clarity, оскільки вони надають критичний візуальний контекст.

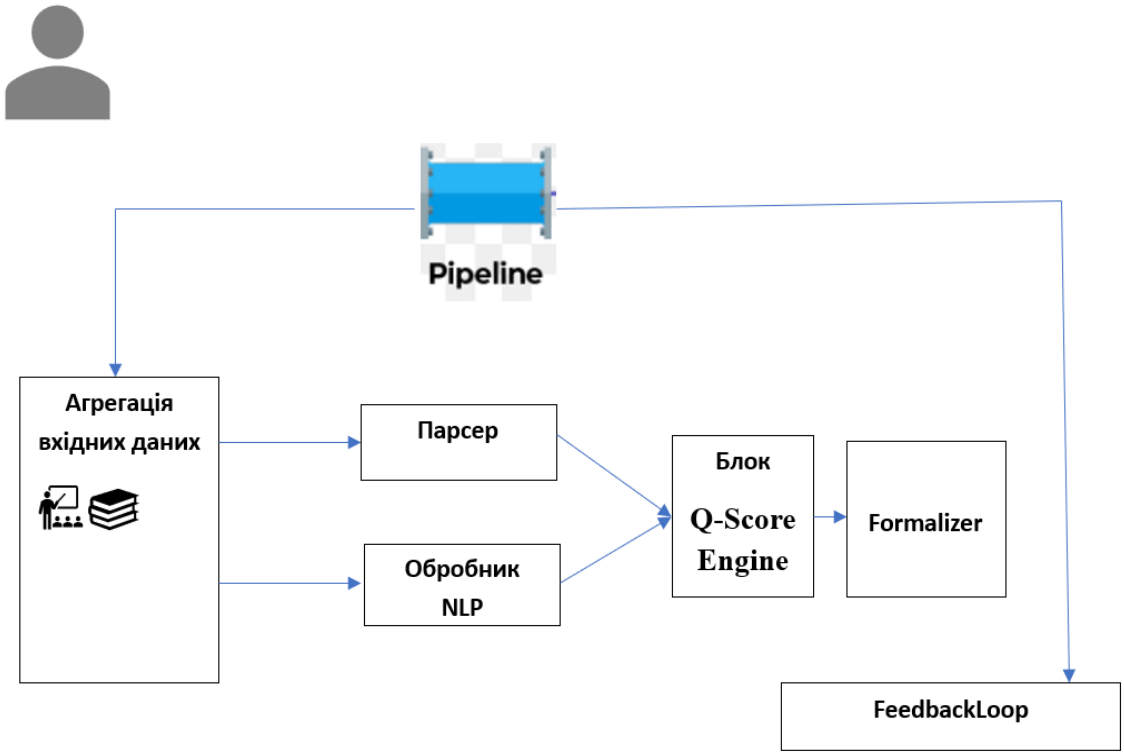


Рисунок 3.1 – Архітектура фреймворку для формалізації вимог до програмного забезпечення на основі оцінки відповідей зацікавлених сторін

Отже, архітектурне рішення полягає у тому, що за основу взято так званий стиль pipeline, тобто послідовний конвеєр. Даний конвеєр в свою чергу має чотири основні етапи і доповнюється розширеннями для забезпечення адаптивності та простежуваності.

3.3. Висновки до 3-го розділу

У третьому розділі визначено вимоги до програмної реалізації системи формалізації вимог до програмного забезпечення. Структуровано функціональні та нефункціональні вимоги, які включають такі пункти, що наведено нижче.

Визначено, що функціональні вимоги формуються для різних блоків і таким чином для модуля інджестування:

– система повинна надавати інтерфейс;

– кожен запит повинен бути асоційований з ідентифікатором зацікавленої сторони;

– забезпечення мультимодальності;

– система повинна проводити попередню обробку.

Для модуля аналізу та скорингу (Q-Score Engine) система повинна використовувати NLP-моделі для аналізу тексту вимоги, зокрема для:

– визначення ключових сутностей;

– лексичного аналізу для виявлення слів-паразитів;

– виявлення кількісних показників.

Нефункціональні вимоги мають забезпечувати:

– продуктивність;

– повний цикл розрахунку Q-Score;

– конфігурованість;

– масштабованість;

– шифрування;

– безпеку;

– юзабіліті.

Також було здійснено проектування архітектури програмного застосунку (фреймворку) для формалізації вимог до програмного забезпечення на основі аналізу якості результатів зацікавлених сторін.

Визначено, що архітектурно фреймворк являє собою послідовний конвеєр (pipeline) з чотирьох основних етапів, доповнений розширеннями для забезпечення адаптивності та простежуваності.

4 ОЦІНКА СИСТЕМИ ДЛЯ ФОРМАЛІЗАЦІЇ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ НА ОСНОВІ ОЦІНКИ ЯКОСТІ РЕЗУЛЬТАТІВ ОПИТУВАННЯ ЗАЦІКАВЛЕНИХ СТОРІН

4.1 Методологія оцінювання програмної системи для формалізації вимог до програмного забезпечення

Дана методологія відноситься до досить складного та цікавого завдання, яке вимагає поєднання бізнес-аналізу, психології комунікації та інженерії програмного забезпечення.

Нижче наведено опис методології для оцінки процесу формалізації вимог до ПЗ, де центральним фільтром і рушієм є оцінювання якості опитування (інтерв'ювання) зацікавлених сторін.

Цю модель можна візуалізувати як багатоступеневу систему очищення та трансформації так званої сирої інформації у чіткі специфікації, де ключовим контрольно-пропускним пунктом є якість проведеного опитування.

Загалом доцільно розглядати формалізація через призму якості опитування. . Основна ідея методології полягає у тому, що традиційні моделі часто розглядають процес лінійно: Збір → Аналіз → Формалізація.

Пропонована модель базується на гіпотезі, що якість формалізованих вимог не може перевищувати якість інформації, отриманої під час опитування, тому процес формалізації починається не після завершення опитування, а під час оцінювання його результатів. Якщо результати опитування тобто сирі дані неякісні, формалізація неможлива або шкідлива і призведе до помилок у програмному забезпеченні.

Ключовими компонентами моделі є п'ять основних блоків і двоє критичних петель зворотного зв'язку.

Блок А: Вхідні дані, тобто власне певна невизначеність, де зацікавлені сторони виступають носіями потреб, проблем та очікувань. При цьому їхнє

розуміння часто неструктуроване, емоційне та суперечливе. Даний блок працює у контексті бізнес-середовища, де є свої обмеження, існуючі системи, правила.

Блок B описує процес безпосереднього опитування, що відноситься до активної фази взаємодії, яка включає не тільки саме інтерв'ю, але й підготовку до нього. В даному блоці відбувається проведення інтерв'ю, анкетування, воркшопи.

Проміжним результатом, тобто Artifact 1 є сирі дані опитування (Raw Elicitation Data). Це можуть бути нотатки, аудіозаписи, транскрипції, заповнені анкети. Це інформація, яка ще не є вимогами.

Блок C включає фільтр якості, що є центральним елементом моделі, тобто це ядро концепції. Тут відбувається не аналіз вимог, а мета-аналіз - аналіз якості отриманої інформації. Сирі дані проходять перевірку за набором критеріїв якості самого опитування.

Фільтр якості включає такі критерії оцінювання якості опитування:

– повнота охоплення, тобто чи були поставлені всі необхідні запитання для конкретної предметної області і чи виявлені так звані білі плями;

– однозначність відповідей, коли визначається чи можна інтерпретувати відповіді стейкхолдера лише одним способом, наприклад, відповідь Система має працювати швидко - це неякісна відповідь, бо швидко не визначено;

– несуперечливість, де визначається чи не суперечить зацікавлена особа сам собі протягом інтерв'ю і чи не суперечать її відповіді відповідям інших ключових стейкхолдерів;

– релевантність визначається визначенням чи стосується отримана інформація цілей проєкту, чи це інформаційний шум;

– рівень довіри та залученості, для визначення якої здійснюється суб'єктивна оцінка аналітика, тобто чи був стейкхолдер відвертим, і чи розумів він питання.

На основі роботи фільтру приймається рішення про достатню якість сирих даних для початку формалізації. Якщо якість визначається, як недостатня, то

активується петля зворотного зв'язку 1, тобто відбувається уточнення. Далі дані повертаються в блок B для проведення додаткових, більш сфокусованих опитувань. Якщо якість визначається як достатня, то дані переходять у блок D.

Блок D відповідає за процес формалізації та відповідно трансформації. Тільки якісні, перевірені сирі дані потрапляють сюди. Тут відбувається переклад з людської мови на інженерну. Саме у даному блоці відбувається моделювання процесів (BPMN, UML), написання User Stories, створення Use Cases, визначення атрибутів якості, тобто визначення нефункціональних вимог.

Проміжний результат роботи даного блоку, або Artifact 2 є чернетка формалізованих вимог (Draft Formal Specifications).

Блок E здійснює валідацію та верифікацію, коли формалізовані вимоги повертаються зацікавленим сторонам для підтвердження. Тут перевіряється, чи правильно ми зрозуміли їхні якісні відповіді. На даному етапі відбувається рецензування специфікацій, демонстрація прототипів, отримання підписів. У цьому блоці приймається рішення про відповідність формалізованим вимогам реальним потребам. Якщо формалізовані вимоги не відповідають реальним потребам, тобто є помилка інтерпретації, то активується петля зворотного зв'язку 2, тобто корекція і здійснюється повернення до Блоку D для виправлення моделей та текстів. Якщо у даній петлі є невідповідність, тобто виявлено нову інформацію, то відбувається повернення аж до Блоку B. Якщо визначається відповідність вимог, то відбувається перехід до фіналу.

На стадії фіналу отримуються вихідні дані відповідно до базової лінії вимог (Requirements Baseline), тобто затверджений, формалізований, якісний набір вимог, готовий до передачі розробникам.

У цій методології аналітик виступає не просто як записувач, а як діагност. Аналітик проводить інтерв'ю зі стейкхолдером щодо нової функції звітності. Наприклад, стейкхолдер каже про те, що він хоче, щоб звіти були гнучкими, відповідно спрацьовує фільтр якості і аналітик розуміє, що відповідь не

відповідає критерію однозначності. Таким чином стає зрозуміло, що це неякісна сировина для формалізації.

Наприклад, замість того, щоб писати вимогу, що система має бути гнучкою, аналітик ставить уточнюючі запитання, тобто повертається до попереднього блоку: Що ви маєте на увазі під гнучкістю? Можливість вибирати колонки? Фільтрувати за датою? Змінювати порядок?.

Тільки отримавши чіткі відповіді, тобто пройшовши фільтр поточного блоку аналітик переходить до наступного блоку і формалізує це як набір історії користувача про фільтрацію та сортування.

Перевагами такої методології є:

1. Раннє виявлення ризиків, оскільки проблеми виявляються на етапі розмови, а не на етапі тестування готового продукту.

2. Економія ресурсів, бо не витрачається час на формалізацію, наприклад, створення діаграм, специфікацій, тобто того, що було неправильно зрозуміло від початку.

3. Підвищення культури комунікації, тому що зацікавлені сторони з часом вчаться давати більш якісні відповіді, розуміючи, що абстрактні побажання не проходять фільтр.

4. Фокус на корені проблеми, оскільки модель змушує аналітика постійно оцінювати не тільки що сказав клієнт, а й наскільки якісно він це сказав.

4.2 Результати роботи системи для формалізації вимог до програмного забезпечення

Оцінювання ефективності програмної системи, призначеної для автоматизованої або напівавтоматизованої формалізації вимог на основі даних

опитувань, вимагає комплексного підходу, що охоплює як процесні показники, так і якість кінцевого результату.

Першочергово варто звернути увагу на метрики, що характеризують рівень автоматизації та економію часу. Ключовим показником тут може слугувати Коефіцієнт автоматичної генерації вимог, який визначає відсоткове співвідношення елементів специфікації, створених системою без втручання людини, до загального обсягу фінального документа вимог. Тісно пов’язаним із цим є показник Скорочення середнього часу на підготовку чернетки специфікації, який розраховується шляхом порівняння часу, витраченого на формалізацію з використанням системи, з історичними даними або контрольними замірами виконання аналогічного завдання вручну бізнес-аналітиком.

Критично важливим блоком є метрики якості перетворення вхідних сирих даних опитувань у формалізовані артефакти. Центральне місце тут посідає семантична точність інтерпретації, яка визначається через експертний аудит вибірки результатів і показує відсоток випадків, коли система коректно зберегла первинний намір зацікавленої сторони без викривлень змісту під час перекладу у формальну мову. Додатково слід оцінювати Здатність до виявлення неоднозначностей, вимірюючи відсоток успішно ідентифікованих системою суперечливих або нечітких відповідей респондентів, що потребують уточнення, порівняно із загальною кількістю таких проблемних місць, знайдених експертами.

Третя група метрик стосується зручності використання результатів роботи системи та обсягу необхідного доопрацювання. Показовим є метрика зусиль на пост-редагування, яку можна виразити через середню кількість правок або людино-годин, які аналітик змушений витратити на доведення згенерованого системою документа до прийнятного стандарту якості. Високий показник зусиль на редагування може свідчити про низьку практичну цінність автоматизації.

Нарешті, для оцінки довгострокового впливу впровадження такої системи варто відстежувати метрики впливу на подальший життєвий цикл розробки, зокрема Зниження щільності дефектів, пов’язаних із вимогами, на етапах тестування та приймання, що свідчитиме про підвищення якості початкових специфікацій завдяки використанню системи.

Оцінювання та перевірку роботи розроблюваного прототип фреймворку ReqQual можна здійснити через симуляцію кількох спринтів, завантаживши в нього 100 сирих запитів від команди зацікавлених сторін.

У таблиці один це продемонстровано. Найбільший виграш – це 80% економії часу бізнес-аналітиків. Замість того, щоб годинами з'ясовувати, що означає зручно, вони лише валідують 45 вже відфільтрованих і структурованих запитів. 38% невдалих відповідей було відхилено автоматично.

Таблиця 4.1 – Оцінювання та перевірка роботи фреймворку

Метрика	**До ReqQual (100 запитів)**	**Після ReqQual (100 запитів)**	**Зміна**
Запити, що потрапили в беклог	100 (100%)	45 (45%)	-55%
Автоматично відхилено (Q-Score < 0.6)	0 (0%)	38 (38%)	+38%
Потребують ручної перевірки (повернено)	100 (100%)	17 (17%)	-83%
Середній час аналітика на очищення 1 вимоги	~15 хв.	~3 хв. (лише валідація)	-80%

На рисунках 4.1 та 4.2 показано, що із 15 сирих запитів, що надійшли до системи було відібрано Q-Score<0,6 і тепер аналітик чи аналітична команда може сфокусуватися не на розшифровці, а на сортуванні зрозумілих завдань за пріоритетом.

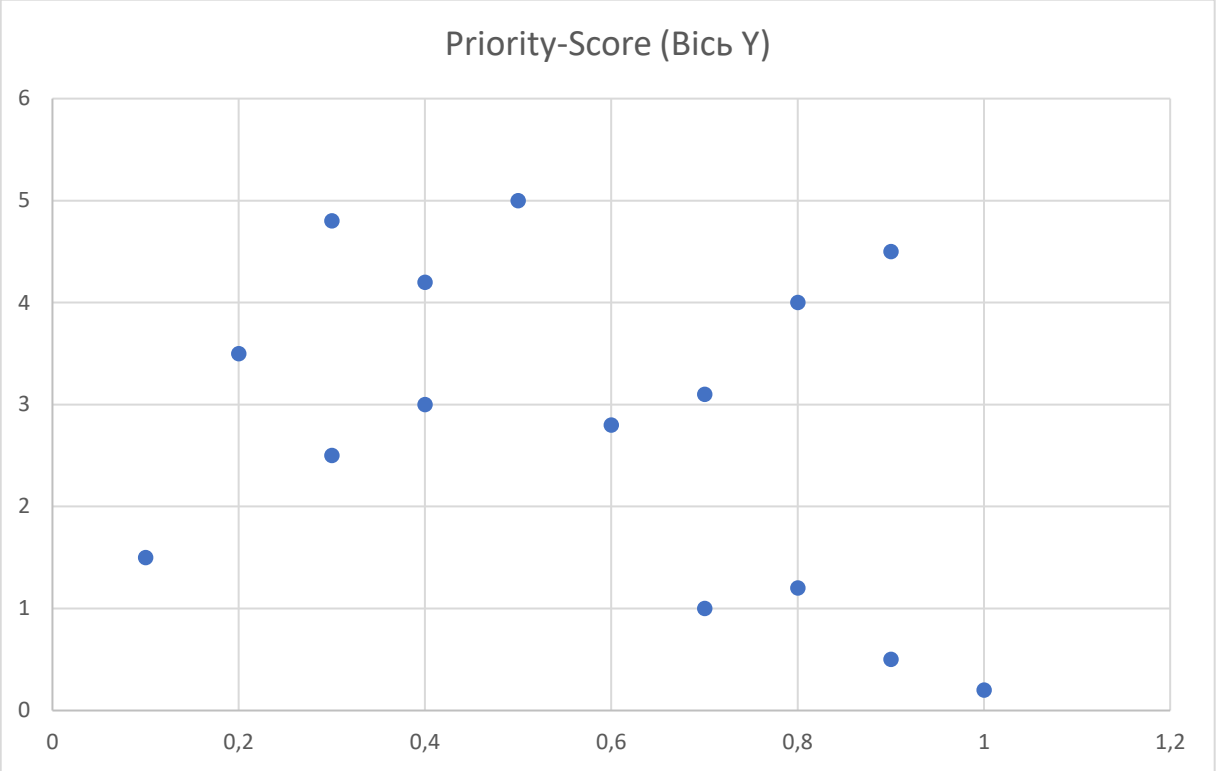


Рисунок 4.1 – Результат

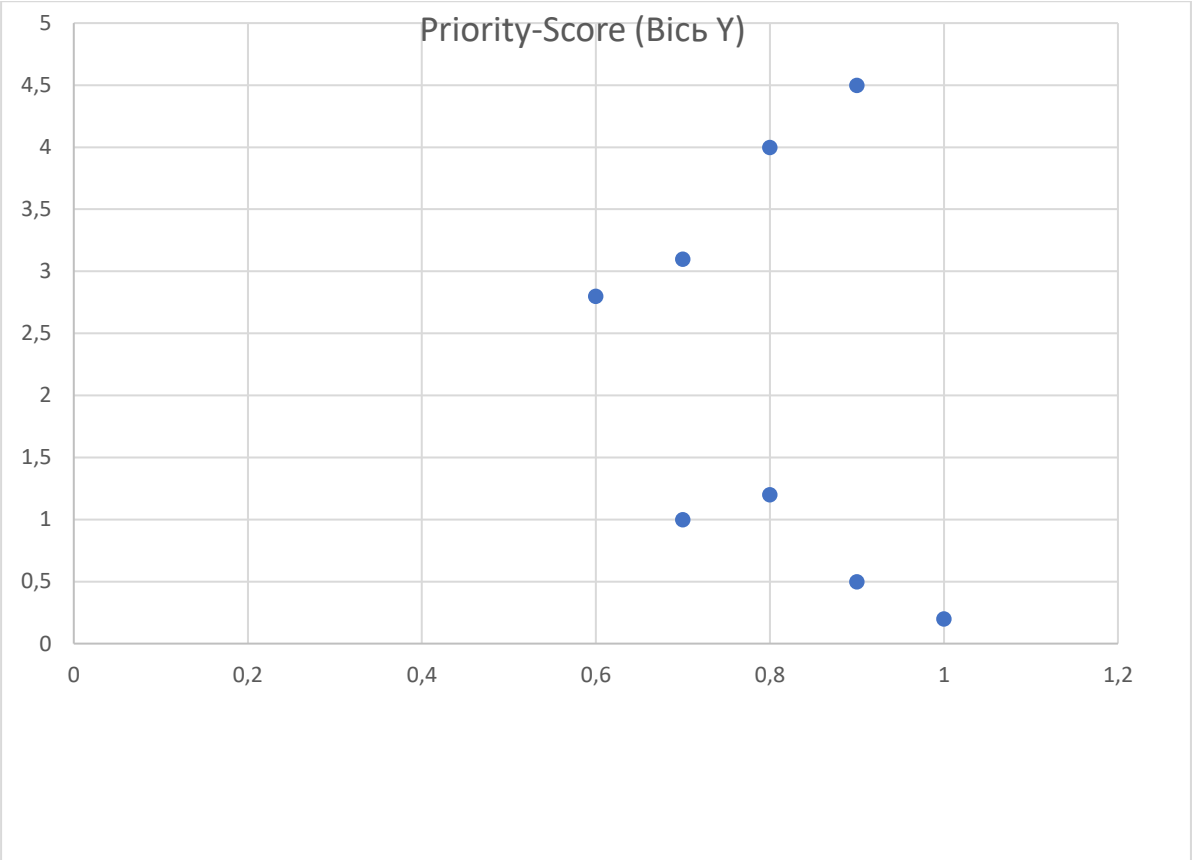


Рисунок 4.2 – Результат оцінки якості програмної системи

Отже, як бачимо деталізація метрик для оцінювання програмної системи, що автоматизує процес формалізації вимог на основі даних опитувань, вимагає заглиблення в методи розрахунку та інтерпретації показників ефективності, якості та впливу, зберігаючи при цьому зв'язний виклад без використання переліків.

4.3 Висновки до 4-го розділу

У розділі було проаналізовано методологію для оцінки роботи програмної системи, а саме фреймворку для формалізації вимог до програмного забезпечення на основі аналізу відповідей зацікавлених сторін.

Оцінювання ефективності роботи такої системи починається з аналізу ступеня автоматизації рутинних задач. Ключовим показником тут є Коефіцієнт автоматичної генерації артефактів вимог, який розраховується як відношення кількості елементів специфікації (наприклад, окремих User Stories, кроків у Use Cases або визначень у глосарії), створених системою повністю автономно і прийнятих аналітиком без змін, до загальної кількості елементів у фінальному документі. Високе значення цього коефіцієнта свідчить про зрілість алгоритмів системи. Доповнює цей показник метрика Відносного скорочення часу циклу формалізації, яка визначається шляхом порівняння середнього часу, витраченого на перетворення сирих транскриптів інтерв'ю в першу чернетку структурованих вимог за допомогою системи, з еталонним часом виконання аналогічного обсягу робіт вручну кваліфікованим бізнес-аналітиком, що дозволяє обчислити економічну доцільність впровадження інструменту.

Критичний блок метрик стосується якості інтелектуальної обробки природної мови, оскільки ціна помилки на цьому етапі є найвищою. Найважливішою тут є Семантична точність інтерпретації намірів, яка вимагає ручного аудиту вибірки результатів експертами для визначення відсотка випадків, коли система коректно ідентифікувала головну бізнес-потребу, обмеження та умови в тексті опитування і трансформувала їх у формальну вимогу без втрати чи викривлення первинного змісту. Оскільки концептуальна модель наголошує на важливості фільтрації неякісних вхідних даних, критичною метрикою є повнота та точність виявлення неоднозначностей. Повнота показує, яку частку об'єктивно нечітких або суперечливих висловлювань стейкхолдерів система змогла автоматично помітити і позначити як такі, що потребують уточнення, порівняно із загальною кількістю проблемних місць, знайдених експертом у тому ж наборі даних. Точність же відображає, скільки із позначених системою як неоднозначних фрагментів дійсно були такими, а не виявилися хибними спрацьовуваннями на зрозумілих реченнях.

Третя група метрик фокусується на взаємодії людини з системою та обсязі необхідних коригувань, що напряму впливає на задоволеність користувачів-аналітиків. Центральним показником тут є індекс зусиль, який можна виміряти як середню кількість атомарних операцій редагування (вставок, видалень, замін слів або переміщень блоків тексту), які аналітик змушений виконати над згенерованою системою чернеткою, щоб довести її до затвердженого стандарту якості; альтернативним виміром може бути час, витрачений на редагування, нормований на обсяг документа.

Нарешті, для оцінки стратегічного впливу системи на весь проєкт використовують відкладені метрики якості кінцевого продукту, зокрема Зниження щільності дефектів категорії вимоги на етапах тестування. Цей показник відстежує кількість багів, виявлених під час системного тестування або приймального тестування користувачами, першопричиною яких було визнано неправильно сформульовану, пропущену або двозначну вимогу, порівнюючи ці дані з аналогічними показниками на проєктах, де формалізація відбувалася вручну. Статистично значуще зниження таких дефектів є найкращим підтвердженням того, що система не просто прискорює процес написання документів, а й покращує їхню сутнісну якість.

ВИСНОВКИ

Отже, в результати проведеного дослідження можна зробити такі висновки.

Перший розділ містить аналітичну складову дослідження та структуризації предметної області, а саме увагу приділено формалізації вимог до програмного забезпечення. В результаті такого дослідження було структуровано поняття вимог до програмного забезпечення, а також визначено важливість та доцільність стандартизації цих вимог та їх впливу на подальшу розробку програмного забезпечення.

Також здійснено аналіз методів та засобів, що допомагають стандартизувати та формалізувати вимоги до програмного забезпечення впродовж всього життєвого циклу програмного продукту.

Окреслено мету роботи, що полягає в удосконаленні методу формалізації вимог до програмного забезпечення із використанням оцінювання якості результатів опитування зацікавлених сторін. Відповідно до мети були встановлені задачі для дослідження.

У другому розділі було здійснено концептуальне моделювання, Запропонована концептуальна модель процесу формалізації вимог до програмного забезпечення, що відходить від традиційного лінійного підходу і ставить у центр уваги критичне оцінювання якості комунікації із зацікавленими сторонами. В основі цієї моделі лежить гіпотеза, що якість кінцевих формалізованих специфікацій не може перевищувати якість «сирої» інформації, отриманої під час усних чи письмових опитувань, тому процес формалізації має починатися не після завершення інтерв'ю, а інтегруватися безпосередньо в процес його оцінювання.

Початковий етап моделі невизначений, де носіями потреб є зацікавлені сторони, чиє розуміння проблем часто буває неструктурованим, емоційним або суперечливим через складний бізнес-контекст. Активна фаза моделі розпочинається з процесу опитування, тобто безпосереднього проведення

інтерв'ю, воркшопів та анкетування, результатом чого стають сирі дані опитування у вигляді нотаток, аудіозаписів та транскрипцій, які ще не є готовими вимогами.

Центральним елементом і своєрідним рушієм усієї моделі виступає фільтр якості отриманої інформації. Перш ніж перейти до будь-якої формалізації, сирі дані піддаються ретельному мета-аналізу, де оцінюється не стільки зміст бізнес-потреби, скільки якість проведеного опитування. Аналітик перевіряє отримані відповіді на відповідність низці критичних критеріїв, визначаючи, чи була забезпечена повнота охоплення предметної області і чи не залишилося білих плям. Критично важливою є перевірка на однозначність, адже відповіді, що допускають множинні інтерпретації, вважаються неякісною сировиною. Також дані аналізуються на предмет внутрішньої несуперечливості, релевантності цілям проєкту та загального рівня довіри до джерела інформації.

У цей момент настає ключова точка прийняття рішення, яка визначає подальший рух процесу. Якщо сирі дані не проходять фільтр якості через неоднозначність або неповноту, процес формалізації блокується, і активується перша петля зворотного зв'язку, що повертає аналітика до етапу опитування для проведення уточнюючих, більш сфокусованих інтерв'ю. Лише після того, як інформація досягає необхідного рівня якості та однозначності, вона допускається до етапу трансформації.

На етапі формалізації якісні, перевірені сирі дані перекладаються з природної мови на інженерну, перетворюючись на моделі процесів, структуровані історії користувачів, варіанти використання та чітко визначені атрибути якості, формуючи чернетку специфікацій. Завершальний цикл моделі передбачає валідацію та верифікацію цих чернеток зацікавленими сторонами, щоб підтвердити правильність інтерпретації їхніх слів. У разі виявлення помилок інтерпретації спрацьовує друга, коротша петля зворотного зв'язку для корекції моделей, і лише після успішного підтвердження формується фінальна базова лінія вимог, готова до передачі в розробку. Таким чином, ця модель

змушує аналітика діяти як діагност, постійно оцінюючи не тільки те, що говорять клієнти, а й те, наскільки якісно вони це роблять, що дозволяє виявляти ризики на найбільш ранніх стадіях проєкту.

У третьому розділі вимоги до програмної реалізації системи формалізації вимог до програмного забезпечення. Структуровано функціональні та нефункціональні вимоги, які включають такі пункти, що наведено нижче.

Визначено, що функціональні вимоги формуються для різних блоків і таким чином для модуля інджестування:

– система повинна надавати інтерфейс;

– кожен запит повинен бути асоційований з ідентифікатором зацікавленої сторони;

– забезпечення мультимодальності;

– система повинна проводити попередню обробку.

Для модуля аналізу та скорингу (Q-Score Engine) система повинна використовувати NLP-моделі для аналізу тексту вимоги, зокрема для:

– визначення ключових сутностей;

– лексичного аналізу для виявлення слів-паразитів;

– виявлення кількісних показників.

Нефункціональні вимоги мають забезпечувати:

– продуктивність;

– повний цикл розрахунку Q-Score;

– конфігурованість;

– масштабованість;

– шифрування;

– безпеку;

– юзабіліті.

Також було здійснено проектування архітектури програмного застосунку (фреймворку) для формалізації вимог до програмного забезпечення на основі аналізу якості результатів зацікавлених сторін.

Визначено, що архітектурно фреймворк являє собою послідовний конвеєр (pipeline) з чотирьох основних етапів, доповнений розширеннями для забезпечення адаптивності та простежуваності.

У четвертому розділі було проаналізовано методологію для оцінки роботи програмної системи, а саме фреймворку для формалізації вимог до програмного забезпечення на основі аналізу відповідей зацікавлених сторін.

Оцінювання ефективності роботи такої системи починається з аналізу ступеня автоматизації рутинних задач. Ключовим показником тут є коефіцієнт автоматичної генерації артефактів вимог, який розраховується як відношення кількості елементів специфікації.

В ході проведення даного дослідження використано методи аналізу та синтезу, абстракції, порівняльні критерії.

Відповідно до теми кваліфікаційної роботи опубліковані тези «Аналіз інструментів та засобів формалізації вимог при проектуванні та розробці програмного забезпечення» на конференції «Актуальні проблеми комп’ютерних наук (АПКН-2025)».

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Bozzano M., Bruintjes H., Cimatti A., Katoen J.-P., Noll T. S. Tonetta Compass 3.0. Tools and algorithms for the construction and analysis of systems (TACAS), Springer, Cham, pp. 379-385. 2019.
2. Bos V., Bruintjes H., Tonetta S. Catalogue of system and software properties. Computer safety, reliability, and security, Springer, Cham. pp. 88-101. 2019.
3. Mahmud N., Seceleanu C., Ljungkrantz Resa O.: an ontology-based requirement specification language tailored to automotive systems 10th IEEE int. Symp. On industrial embedded systems. pp. 1-10.
4. Systems and software engineering. Life cycle processes. Requirements engineering: ISO/IEC/IEEE 29148:2018. Geneva: ISO/IEC, 2018. – 102 p.
5. Wiegers K. Software Requirements / K. Wiegers, J. Beatty. - 3rd ed. - Redmond : Microsoft Press, 2013. - 672 p.
6. Lucio L., Rahman S., Cheng C., Mavin A. Just formal enough? automated analysis of EARS requirements. NFM 2017, pp. 427-434.
7. Stachtiari E., Mavridou A., Katsaros P., Bliudze S., Sifakis J.. Early validation of system requirements and design through correctness-by-construction. J Syst Software, 145 (2018), pp. 52-78.
8. Cicchetti, A., Kühne, T., Pierantonio, A. *et al.* Guest editorial for the special section on the 26th international conference on model-driven engineering languages and systems (MODELS 2023). *Softw Syst Model* (2025). https://doi.org/10.1007/s10270-025-01322-0/.
9. The Importance of Technical Support in Software Development Lifecycles - Enhancing Efficiency and Success. URL: https://moldstud.com/articles (Accessed: 01 Oktober 2025).
10. Spencer J. *5C's of critical consuming.* Available at: https://www.youtube.com/watch?v=xf8mjbVRqao (Accessed: 10 Oktober 2025).

11. Classification of Software Requirements - Software Engineering. URL: https://www.geeksforgeeks.org/software-engineering/software-engineering-classification-of-software-requirements/ (Accessed: 01 Oktober 2025).
12. Gerhard Krüger. How to Write Good Software Requirements (with Examples). URL: https://www.perforce.com/blog/alm/how-write-software-requirements-specification-srs-document (Accessed: 02 Oktober 2025).
13. The Guide to Writing Software Requirements Specification Document. URL: https://8allocate.com/blog/the-ultimate-guide-to-writing-software-requirements-specification/ (Accessed: 02 Oktober 2025).
14. What are the types of requirements in software engineering? URL: https://www.techtarget.com/searchsoftwarequality/answer/What-are-requirements-types (Accessed: 02 Oktober 2025).
15. Leong I. T., Barbosa R. Ranslating natural language requirements to formal specifications: A study on gpt and symbolic nlp. *N 2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W*. 2023. P. 259–262.
16. Leong I. T., Barbosa R. Ranslating natural language requirements to formal specifications: A study on gpt and symbolic nlp. *N 2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W*. 2023. P. 259–262.
17. Lano K., Rahimi S., Tehrani S. *et al.* Introduction to theme section on requirements formalisation. *Softw Syst Model*, 1451–1453 (2024). https://doi.org/10.1007/s10270-024-01241-6.
18. Hammer H., Cauwels M., Hertz B. Ntegrating runtime verification into an automated UAS traffic management system. *Innovations in Systems and Software Engineering*. 2022. Т. 18, вип. 1. С. 567–580.
19. Xu J. Y., Wang Y. Formal Software Requirement Elicitation based on Semantic Algebra and Cognitive Computing. *2020 IEEE 19th International Conference on Cognitive Informatics & Cognitive Computing (ICCI*CC)*.

20. The Guide to Writing Software Requirements Specification. URL: https://8allocate.com/blog/the-ultimate-guide-to-writing-software-requirements-specification/ (Accessed: 02 Oktober 2025).
21. Mahmud N., Seceleanu C., Ljungkrantz O. Specification and semantic analysis of embedded systems requirements: from description logic to temporal logic. *Software engineering and formal methods, Springer*. 2017. С. 332–348.
22. Hammer H., Cauwels M., Hertz B. Ntegrating runtime verification into an automated UAS traffic management system.. Nnovations in Systems and Software Engineering. 2022. Т. 18, вип. 1. С. 567–580.
23. A systematic mapping of semi-formal and formal methods in requirements engineering of industrial cyber-physical systems / Z. Farzana та ін. *Journal of Intelligent Manufacturing*. 2022. Т. 33, вип. 6. С. 1603–1638.
24. Software Development as a Formalisation Process. URL: https://antonioritosilva.org/software-engineering-companion/what-is-software-engineering/software-development-as-a-formalisation-process/ (Accessed: 22 Oktober 2025).
25. Guide to the Software Engineering Body of Knowledge (SWEBOK Guide) / eds. P. Bourque, R. E. Fairley. – Version 3.0. – Piscataway : IEEE Computer Society, 2014. – 335 p.

ДОДАТОК А

(обов’язковий)

КОПІЇ НАУКОВИХ ПУБЛІКАЦІЙ

УДК 004.9

Гордієнко Є.О.

Хмельницький національний університет

АНАЛІЗ ІНСТРУМЕНТІВ ТА ЗАСОБІВ ФОРМАЛІЗАЦІЇ ВИМОГ ПРИ ПРОЕКТУВАННІ ТА РОЗРОБЦІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розглянуто задачу формалізації вимог до програмного забезпечення, а також інструменти та засоби, що дозволяють здійснювати правильну та чітку системи формалізації вимог. Здійснено аналіз інструментів та засобів системи формалізації вимог.

The article of formalizing software requirements is considered, as well as the tools and means that allow for the implementation of a correct and clear system of formalizing requirements. An analysis of the tools and means of the system of formalizing requirements is carried out.

Формалізація вимог до програмного забезпечення - це критично важливий етап у життєвому циклі розробки, який полягає у систематичному перетворенні неформальних, часто неоднозначних потреб замовника на чіткі, верифіковані, узгоджені та повні технічні специфікації. Це процес, що переводить розпливчасті ідеї та очікування в точну, структуровану мову, яка може бути зрозуміла та використана як розробниками для кодування, так і тестувальниками для перевірки відповідності.

Основне завдання полягає в усуненні двозначності, оскільки вимоги, виражені природною мовою, неминуче містять термінологічну невизначеність, прогалини в логіці та суперечності. Формалізація використовує методи, що дозволяють перевірити кожну вимогу на узгодженість та відповісти на запитання чи не суперечить вона іншим вимогам та визначити повноту, тобто чи описано всі необхідні функції.

Цей процес охоплює кілька ключових дій. Спочатку проводиться детальний аналіз зібраних вимог для їхньої класифікації на функціональні (що система повинна робити) та нефункціональні (як система повинна працювати, включаючи вимоги до продуктивності, безпеки та надійності). Потім відбувається їхнє структурування із застосуванням стандартних шаблонів та нотацій, таких як використання сценаріїв (Use Cases) або мови UML. На цьому етапі вимоги перетворюються на технічні атрибути, коли кожна функціональна вимога повинна мати унікальний ідентифікатор, бути простежуваною до джерела та, що найважливіше, верифікованою, тобто повинна існувати можливість розробити тест, який підтвердить її виконання.

Кінцевим результатом формалізації є створення документу специфікації вимог до програмного забезпечення (SRS), який слугує єдиним джерелом істини та

контрактом між замовником і командою розробки. Без цієї строгої формалізації проєкт неминуче стикається зі збільшенням вартості, затримками та невідповідністю кінцевого продукту початковим очікуванням.

Загалом, дослідники виділяють декілька інструментів із доступністю поза межами проектів. Також можна виокремити ще один засіб, що доступний як частина набору інструментів COMPASS [1], який спрощує формальну специфікацію вимог у контексті моделей AADL [2].

У [3] дослідники представили інструмент під назвою DODT, розроблений у рамках проекту CESAR [4], що дозволяє проекційне редагування та типізацію вимог на основі шаблонного синтаксису (Requirements Specification Language - RSL), атрибутної онтології та предметно-специфічної онтології. Були реалізовані перевірки валідації на основі онтологій, які дозволяють виявляти суперечності шляхом попарного порівняння вимог, іменників, які не визначені в онтології тощо. Хоча DODT не реалізує точного зв'язку шаблонів зі шаблонами властивостей, однак авторами надаються пропозиції щодо шаблонів, які можуть бути придатними для опису заданої вимоги.

Інструмент EARS-CTRL [5] був введений для забезпечення шляхом побудови коректності вимог, написаних з використанням шаблонів Easy Approach to Requirements Syntax (EARS). Інструмент перевіряє, чи можна синтезувати контролер з набору вимог. Якщо контролер неможливо синтезувати, то існують конфліктуючі вимоги. EARS-CTRL дозволяє редагувати проекційні вимоги на основі глосарію, визначеного для області синтезу контролерів. Вимоги аналізуються як формули LTL. Ефективність аналізу залежить від визначеної користувачем семантичної інформації (наприклад, простих предикатів) для заданого глосарію.

Інструмент RERD [6] підтримує специфікацію вимог та виведення властивостей на основі шаблонів та шаблонів властивостей, а також поступове конструювання систем для реалізації похідних властивостей за допомогою підходу проектування «знизу вгору». Під час редагування вимог користувач взаємодіє з базовими онтологіями, запитуючи їх та отримуючи доступ до екземплярів їхніх класів для пошуку відповідних термінів. Таким чином, словник шаблонних атрибутів обмежений термінами, які можна ідентифікувати в рамках предметно-специфічних онтологій та онтологій, специфічних для проектованої системи. Зв'язки шаблонних атрибутів відіграють ключову роль у семантичному аналізі та виведенні властивостей, де кожен шаблон пов'язаний із заздалегідь визначеними шаблонами властивостей, які можуть формально його фіксувати. Для забезпечення дотримання властивостей користувач може вибрати один із доступних шаблонів архітектури та параметризувати їх, вибираючи компоненти з поступово побудованої моделі системи. Потім необхідно перевірити відсутність взаємоблокувань у результуючій моделі, що пропонується зробити за допомогою спеціального інструменту D-Finder. Для властивостей, які неможливо забезпечити за допомогою існуючих шаблонів архітектури, користувачеві доведеться використовувати зовнішні інструменти, такі як перевірка моделей.

Щодо підходу на основі COMPASS, у [1] дослідники представили інший підхід, згідно з яким формальні властивості не створюються з шаблонів шляхом заміни їхніх заповнювачів станами/подіями, пов'язаними з атрибутами, визначеними шаблоном, а шляхом присвоєння значень атрибутам системної моделі. Більш конкретно, було введено набір властивостей, які дозволяють пов'язувати значення з певними елементами моделі AADL.

Формальна семантика системних властивостей спирається на поведінкову семантику мови SLIM, тоді як логіка, що використовується для визначення більшості з них, є варіантом метричної часової логіки (MTL). Хоча таксономія формальних властивостей порівняно обмежена у виразності, вона повністю усуває необхідність вибору шаблону та винаходження екземпляра. Формальні властивості можна аналізувати на узгодженість між різними рівнями абстракції системної моделі або використовувати як припущення чи гарантії компонентів для проектування на основі контрактів. Для останньої мети COMPASS вимагає додаткової специфікації уточнення контракту, яке пов'язує контракт з його підкомпонентів. Це дозволяє виконувати різні аналізи, такі як узгодженість або наслідки для будь-якої підмножини властивостей контракту, щоб перевірити правильність уточнень контракту, а також звузити контракт таким чином, щоб уточнення залишалося чинним.

Отже, специфікація вимог може базуватися на онтологіях предметної області, які використовуються для виявлення відсутньої інформації та усунення невідповідностей і випадків недостатньої специфікації. Виведення властивостей залежить від формальної моделі системи, яка може бути поступово побудована за допомогою компонентного процесу проектування.

Перелік посилань

1. M. Bozzano, H. Bruintjes, A. Cimatti, J.-P. Katoen, T. Noll, S. Tonetta Compass 3.0. Tools and algorithms for the construction and analysis of systems (TACAS), Springer, Cham (2019), pp. 379-385.
2. V. Bos, H. Bruintjes, S. Tonetta. Catalogue of system and software properties. Computer safety, reliability, and security, Springer, Cham (2016), pp. 88-101.
3. N. Mahmud, C. Seceleanu, O. Ljungkrantz Resa : an ontology-based requirement specification language tailored to automotive systems 10th IEEE int. Symp. On industrial embedded systems (2015), pp. 1-10.
4. Rajan, T. Wahl CESAR-cost-efficient methods and processes for safety-relevant embedded systems Springer (2013).
5. L. Lucio, S. Rahman, C. Cheng, A. Mavin. Just formal enough? automated analysis of EARS requirements. NFM 2017 (2017), pp. 427-434. NASA formal methods - 9th int. Symp.
6. E. Stachtiari, A. Mavridou, P. Katsaros, S. Bliudze, J. Sifakis. Early validation of system requirements and design through correctness-by-construction. J Syst Software, 145 (2018), pp. 52-78.

ДОДАТОК Б

(обов’язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ

Кафедра інженерії програмного забезпечення

Метод формалізації вимог до програмного забезпечення на основі оцінки якості результатів опитування зацікавлених сторін

Автор роботи:

студент групи ІПЗм-24-1 Гордієнко Є. О.

Керівник роботи:

кандидат педагогічних наук, доцент Онишко О.Г.

Актуальність теми

Потреба стандартизації процесів інженерії програмного забезпечення. Неформальне подання вимог часто призводить до їх непослідовності, неоднозначності та неповноти, що ускладнює подальшу розробку.

Формалізація вимог дозволяє забезпечити їх чіткість і структурованість. Моделювання процесів розробки ПЗ є важливим підходом для фіксації й оптимізації цих процесів.

Використання методів формалізації, зокрема на основі оцінки якості результатів опитувань зацікавлених сторін, є актуальним і необхідним етапом під час визначення вимог у життєвому циклі програмного продукту.

Мета і завдання дослідження

Мета: удосконалення методу формалізації вимог до програмного забезпечення із використанням оцінювання якості результатів опитування зацікавлених сторін.

Завдання дослідження:

1. Виконати аналіз предметної області та підходів до визначення вимог до ПЗ.
2. Здійснити огляд і порівняння існуючих методів формалізації вимог.
3. Удосконалити метод формалізації з урахуванням оцінки якості результатів опитувань зацікавлених сторін
4. Розробити алгоритм роботи системи формалізації вимог на основі даних опитувань.

Об’єкт та предмет дослідження

Об’єкт дослідження: процес формалізації вимог до програмного забезпечення на етапі аналізу вимог до розроблюваного програмного продукту.

Предмет дослідження: методи формалізації вимог до програмного забезпечення на етапі аналізу вимог до розроблюваного програмного продукту.

Наукова новизна

1. Удосконалення методу формалізації вимог до програмного забезпечення із використанням оцінювання якості результатів опитування зацікавлених сторін.

2. Розроблено алгоритм роботи системи формалізації вимог до програмного забезпечення із використанням оцінювання якості результатів опитування зацікавлених сторін.

Практична значимість

Результати кваліфікаційної роботи можуть бути використані розробниками програмного забезпечення під час аналітичного етапу формування вимог і технічного завдання.

Удосконалений метод формалізації вимог на основі оцінки якості опитувань зацікавлених осіб підвищує якість документації та зменшує ризики помилок у розробці програмних продуктів.

1 РОЗДІЛ Аналіз предметної області

Проблематика формалізації вимог активно досліджується в науковій спільноті.

Зокрема, праці **Басіра, Салама, Фрідріха** присвячені структурованості вимог;

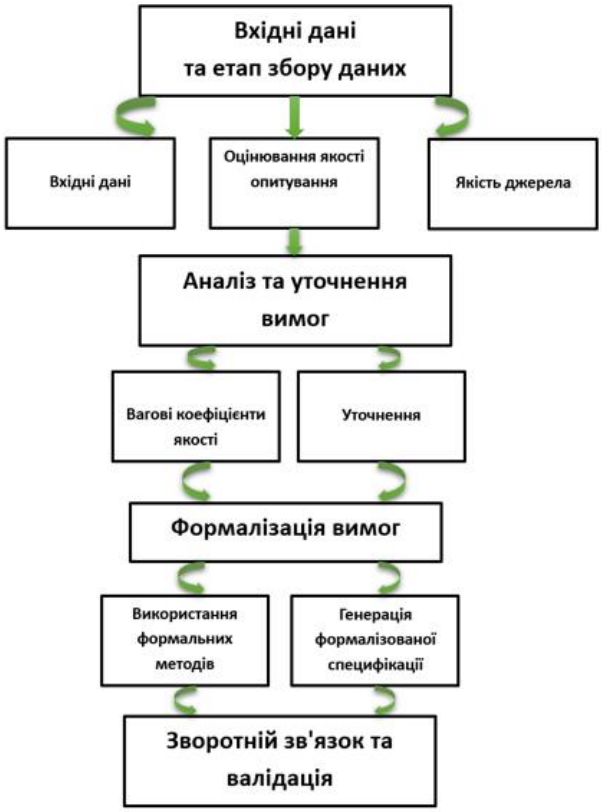
Цзян-піна та співавторів — класифікації й виявленню неявних вимог;

Сатклiффа і Сойєра — методам виявлення вимог;

Леффігвелла та Відріга — дослідженню неявних знань зацікавлених сторін.

Це підкреслює актуальність обраного напряму дослідження

2 РОЗДІЛ Концептуальна модель

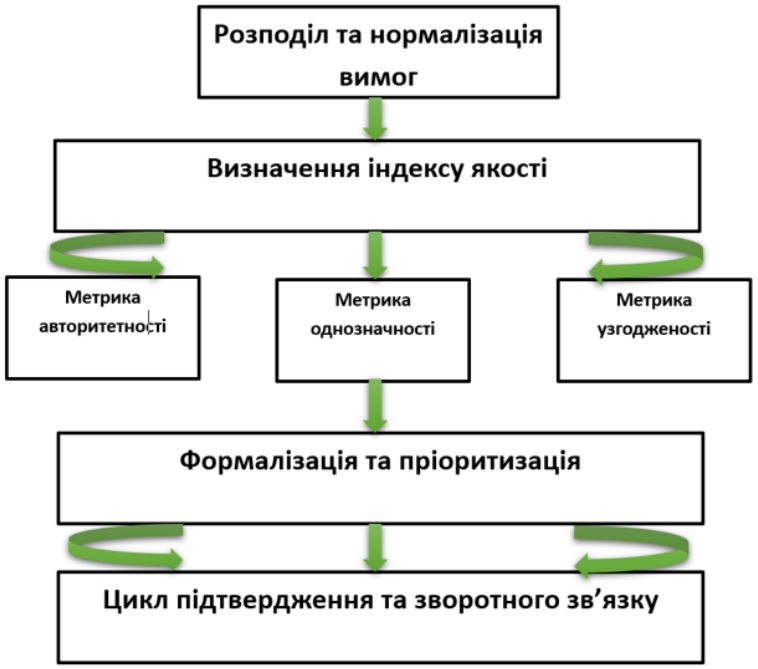


2 РОЗДІЛ Метод формалізації вимог до програмного забезпечення із використанням оцінювання якості результатів опитування зацікавлених сторін

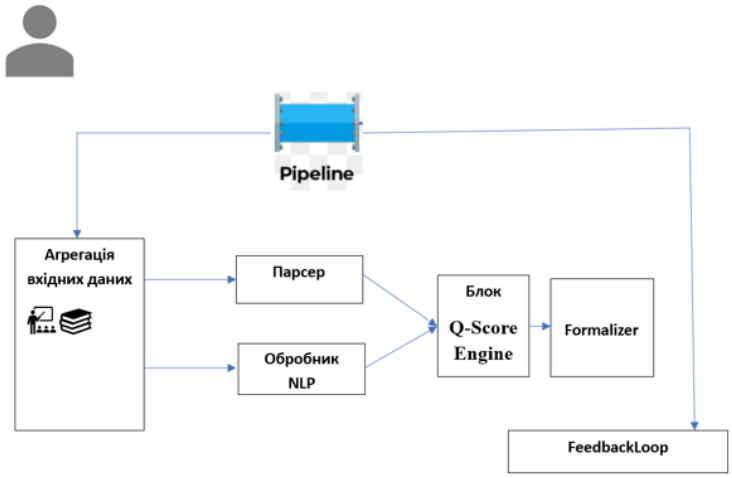
Метод формалізації вимог до програмного забезпечення із використанням оцінювання якості результатів опитування зацікавлених сторін передбачає введення та подальшу обробку неявних вимог зацікавлених сторін.

Він ґрунтується на багатокритеріальному аналізі рішень (MCDA), який дає змогу обирати найкращу альтернативу з урахуванням оцінки за кількома, часто конфліктуючими, критеріями

2 РОЗДІЛ Алгоритм формалізації вимог до ПЗ на основі оцінювання якості результатів опитування зацікавлених сторін



3 РОЗДІЛ Архітектура програмної системи



4 РОЗДІЛ Результати

Таблиця 1 - Ефективність фільтрації (Q-Score)

Метрика	До ReqQual (100 запитів)	Після ReqQual (100 запитів)	Зміна
Запити, що потрапили в беклог	100 (100%)	45 (45%)	-55%
Автоматично відхилено (Q-Score < 0.6)	0 (0%)	38 (38%)	+3800%
Потребують ручної перевірки (повернено)	100 (100%)	17 (17%)	-83%
Середній час аналітика на "очистку" 1 вимоги	~15 хв.	~3 хв. (лише валідація)	-80%

4 РОЗДІЛ Результати

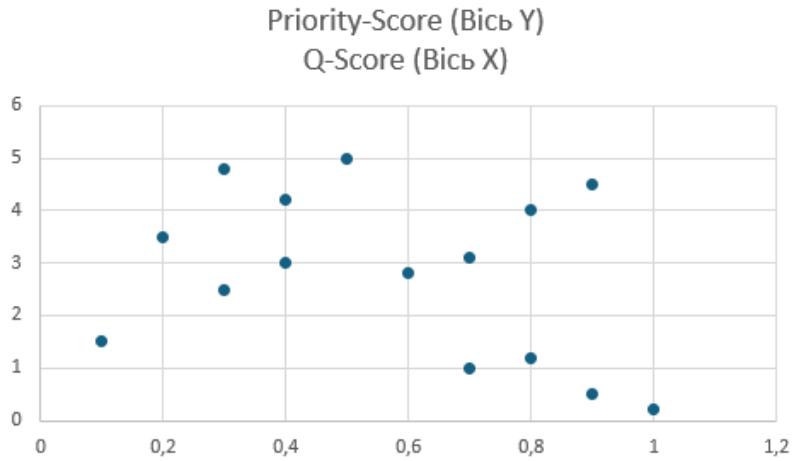


Рисунок 1

4 РОЗДІЛ Результати

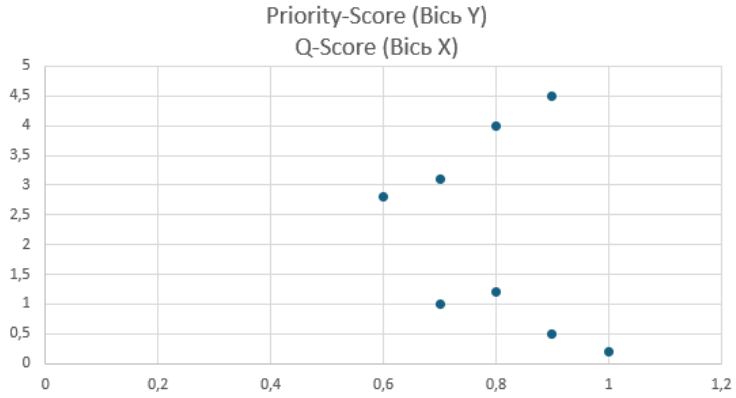


Рисунок 2

Публікації за темою роботи

Гордієнко Є.О. АНАЛІЗ ІНСТРУМЕНТІВ ТА ЗАСОБІВ ФОРМАЛІЗАЦІЇ ВИМОГ ПРИ ПРОЕКТУВАННІ ТА РОЗРОБЦІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ. Збірник наукових праць за матеріалами XVII Всеукраїнської науково-практичної конференції «Актуальні проблеми комп'ютерних наук АПКН-2025», 14-15 листопада 2025, Хмельницький 2025, стор.89-91

Висновки:

Мети роботи досягнуто:

1. Проведено удосконалення методу формалізації вимог до програмного забезпечення із використанням оцінювання якості результатів опитування зацікавлених сторін.
2. Розроблено алгоритм роботи системи системи формалізації вимог до програмного забезпечення із використанням оцінювання якості результатів опитування зацікавлених сторін.
3. Створено фреймворк на основі удосконаленого методу.
4. Результати експериментального дослідження свідчать про ефективність даного фреймворку.

Дякую за увагу!

СУПРОВІДНІ ДОКУМЕНТИ

Завідувачу кафедри
інженерії програмного забезпечення
проф. Леоніду БЕДРАТЮКУ
студента групи ІПЗм-24-1
Бордієнко Є.О
Ім'я, ПРІЗВИЩЕ

ЗАЯВА

Прошу закріпити за мною тему кваліфікаційної роботи освітнього ступеня «магістр» за спеціальністю 121 «Інженерія програмного забезпечення»:
Метод формалізації вимог до програмного забезпечення на основі оцінки якості результатів аналізування текстових історій
(керівник кваліфікаційної роботи – Онищко Оксана Григорівна)
Ім'я, ПРІЗВИЩЕ

01.09.2025
Дата

Підпис здобувача

Завідувачу кафедри інженерії програмного
забезпечення проф. Леоніду БЕДРАТЮКУ
здобувача вищої освіти
Гордієнка Євгенія Олеговича
факультет ІТ, 2 курс, група ІПЗм-24-1

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності в Хмельницькому національному університеті, згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщений, та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії Хмельницького національного університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-обчислювального комплексу StrikePlagiarism та/або програмно-технічного засобу AntiPlagiarism і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення текстових збігів у роботах.

Робота надається для перевірки в електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

01.09.2025
дата

підпис

Anti-Plagiarism (UA) v-16.693

The maximum coincidence with one document 16.0%

Dictionaries check: UA, US, RU. **Errors in the documents: 24%**

ID: 252780 Title: МКР_Метод формалізації вимог до програмного забезпечення на основі оцінки якості результатів опитування зацікавлених сторін Added in a DB: 2025-12-14 Authors: Євгеній ГОРДІЄНКО Heads: канд. пед. наук, доцент Оксана ОНИШКО Consultants: Opponents:	Document		Sum coincidence on the DB	
	Symbols	Lexemes	Symbols	Lexemes
	103769	676	17061 (16%)	121 (18%)

Plagiarism sources

ID	Description	Plagiarism presence in the document	
		Symbols	Lexemes
249054	Title: Переддипломна практика Added in a DB: 2025-10-14 Authors: Євген ГОРДІЄНКО Heads: Оксана ОНИШКО, канд. пед.наук, доцент Consultants: Opponents:	16250 (16.0%)	120 (18.0%)

Протокол аналізу звіту подібності експертом

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Євгеній ГОРДІЄНКО

Співавтор:

Назва: Метод формалізації вимог до програмного забезпечення на основі оцінки якості результатів опитування зацікавлених сторін

Експерт: канд. пед. наук, доцент Оксана ОНИШКО

Підрозділ: Кафедра інженерії програмного забезпечення

Коефіцієнт подібності 1:0.6%

Коефіцієнт подібності 2:0%

Мікропробіли: 13

Заміна букв: 1

Інтервали: 0

Білі знаки: 0

Дата створення звіту: 2025-12-14 19:16:28.0

Після аналізу Звіту подібності констатую наступне:

☑ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

Дата 14.12.2025 експерт (Оксана О.В.)

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатами звіту/звітів перевірки роботи, продукованими програмно-технічним засобом (ами), на наявність текстових збігів.

Назва кваліфікаційної роботи: «Метод формалізації вимог до програмного забезпечення на основі оцінки якості результатів опитування зацікавлених сторін»

Автор: Гордієнко Євгеній Олегович

Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»

Спеціальність: 121 – Інженерія програмного забезпечення

Науковий керівник: Онишко Оксана Григорівна, кандидат технічних наук, доцент

Після аналізу звіту/звітів зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається до захисту.	**відповідає**
2	Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована.	
3	Перевищено граничне значення рівня подібностей. Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнуті. Робота може бути допущена до захисту після того, як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Запозичення, виявлені у роботі є законними і не є плагіатом, оскільки:

1) у тексті кваліфікаційної роботи системою перевірки на плагіат StrickePlagiarism виявлено схожість з деякими документами у частині загальновживаних обов'язкових словосполучень у стандартних бланках (титулка, бланк завдання), у структурі змісту, у назвах розділів/підрозділів, у назвах публікацій переліку джерел посилання тощо;

2) в якості запозичень системою StrickePlagiarism було зафіксовано деякі послідовності вихідного коду і посилання на бібліотеки, які є стандартними мовними конструкціями програмування та не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;

3) запозичення, виявлені в тексті роботи, є фрагментарними або мають належним чином оформлені посилання;

4) виявлені модифікації тексту не впливають на відсоток схожості.

Максимальний обсяг запозичень, визначений системою Anti-Plagiarism, складає 16 %. За системою StrickePlagiarism коефіцієнт подібності 1 складає 0.6%, коефіцієнт подібності 2 складає 0 %.

Дата 02.12.2025

Завідувач кафедри _____ Леонід БЕДРАТЮК

Гарант освітньої програми _____ Оксана ЯШИНА

Керівник кваліфікаційної роботи _____ Оксана Онишко

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
освітнього ступеня «Магістр»

Дипломник Гордієнко Євгеній Олегович

Тема Метод формалізації вимог до програмного забезпечення на основі оцінки якості результатів опитування зацікавлених сторін

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень немає ; кількість сторінок записки 85
1.Короткий зміст пояснювальної записки та прийнятих рішень У кваліфікаційній роботі виконано комплексний аналіз проблеми формалізації вимог до програмного забезпечення, зокрема у частині впливу якості опитування зацікавлених сторін на достовірність та коректність сформованих вимог. Проведено дослідження існуючих методів, моделей та інструментів формалізації, проаналізовано їх сильні та слабкі сторони. На основі проведеного аналізу удосконалено метод формалізації вимог до програмного забезпечення шляхом інтеграції оцінювання якості результатів опитування стейкголдерів. Запропонована модель включає етапи структуризації відповідей зацікавлених сторін, оцінювання їхньої якості, пріоритезації та уточнення вимог. Розроблено алгоритм та архітектуру програмної системи, що реалізує запропонований метод. Продемонстровано результати роботи системи та її придатність до практичного використання у процесі інженерії вимог.

2. Висновок про відповідність роботи поставленому завданню Кваліфікаційна робота відповідає затвердженому завданню, а поставлені цілі досягнуті у повному обсязі.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи
У вступі наведено обґрунтування актуальності, мети, завдань, об'єкта та предмета дослідження.
У першому розділі проведено аналіз предметної області, визначено наукові підходи та сформульовано постановку задачі.
У другому розділі представлено модель та удосконалений метод формалізації вимог.
У третьому розділі описано архітектуру програмної системи.
У четвертому розділі наведено результати реалізації та оцінювання системи.

4. Позитивні сторони роботи
– Ґрунтовний аналіз предметної області.
– Обґрунтований та актуальний удосконалений метод формалізації вимог.
– Реалізація програмного забезпечення, що підтверджує практичну цінність роботи.
– Структурований та логічний виклад матеріалу.
5. Негативні сторони роботи
– окремі частини теоретичного огляду подані узагальнено та потребують глибшого аналізу;
– запропонований метод можна було б більш детально формалізувати;

– результати роботи системи подані стисліше, ніж це необхідно для повної оцінки ефективності;

6. Оцінка графічного оформлення та пояснювальної записки Оформлення виконано акуратно та відповідно до вимог. Графічні матеріали подані коректно, хоча деяким з них не вистачає більш детальних пояснень.

7. Відгук про кваліфікаційну роботу в цілому Робота є завершеним дослідженням, яке демонструє володіння основними методами інженерії вимог та здатність автора застосовувати отримані знання на практиці. Незважаючи на окремі недоліки, робота має певну наукову та практичну цінність

8. Інші зауваження Бажано доопрацювати якість формулювань, логічні переходи між розділами та покращити стиль викладення. Експериментальну частину слід значно поглибити.

9. Оцінка кваліфікаційної роботи З огляду на переваги та недоліки, робота заслуговує оцінки "задовільно".

РЕЦЕНЗЕНТ (прізвище, ім'я, по-батькові, посада, місце роботи) Пилипчук Олександра Геннадіївна, д.т.н., професор, професор кафедри КІІС

"12" 12 2025 р.
(підпис)