БАКАЛАВРСЬКА РОБОТА

БР.ІП – 86.00.00.000 ПЗ

Група ІП-22-3

Ходак Олександр

2026

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ходак Олександр Ігорович

(прізвище, ім'я, по батькові)

УДК 004.942
(індекс)

БАКАЛАВРСЬКА РОБОТА

Методи управління вимогами до програмного забезпечення
(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня _____Ходак О.І._____
(підпис, ініціали та прізвище здобувача)

Науковий керівник _____ Зікратий Сергій Вікторович, доцент_____
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри

_доц._____Бандура В.В.__
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2026

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доцент. В.В. Бандура
“ ____ ” _____2026 р.

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Ходак Олександру Ігоровичу

(прізвище, ім’я, по-батькові)

1. Тема проекту (роботи) "Методи управління вимогами до програмного забезпечення "

керівник проекту (роботи) доц. Зікратий С.В.

затверджені наказом вищого навчального закладу від **“** 21 ” квітня 2026 р. № 179/7

2. Строк подання студентом проекту (роботи) 10 червня 2026 р.

3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

4. Зміст розрахунково - пояснювальної записки(перелік питань, які потрібно розробити)

1. Аналіз вимог до програмного забезпечення
2. Аналіз вимог і постановка задачі
3. Проектування системи
4. Реалізація проекту
5. Тестування та впровадження

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1 Діяльності та їхні основні завдання в рамках управління вимогами (рис. 1.1, ст. 14)

2 Структура управління вимогами (рис. 3.1, ст. 52)

3 Внесок методів управління знаннями у глобальну розробку програмного забезпечення (рис. 3.2, ст. 54)

4 Діаграма сценаріїв використання RM-Tool на загальному рівні (рис. 3.3, ст. 64)

5 Діаграма класів для RM-Tool (рис. 3.4, ст. 66)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

2. Дата видачі завдання **2026 р.**

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту	Примітка
1	Визначення та обгрунтування теми роботи	17.03.2026	виконано
2	Огляд існуючих концепцій, рішень та сервісів в даній області	21.04.2026	виконано
3	Побудова моделі або алгоритму власного рішення	01.05.2026	виконано
4	Документування реалізації власного оригінального рішення вибраними засобами	21.05.2026	виконано
5	Оформлення пояснювальної записки бакалаврської роботи	09.06.2026	виконано

Студент _____Ходак О.І._
(підпис)

Керівник роботи _____Зікратий С.В.
(підпис)

АНОТАЦІЯ

Бакалаврська робота містить 78 сторінок, 6 рисунків, 10 таблиць, список використаних джерел із 40 найменувань.

Метою роботи є дослідження методів управління вимогами до програмного забезпечення, аналіз сучасних підходів до їх виявлення.

Об'єкт дослідження: процес управління вимогами до програмного забезпечення.

Предмет дослідження: методи, моделі та інструменти управління вимогами, що використовуються на різних етапах життєвого циклу ПЗ.

Результати дослідження: проаналізовано сучасні підходи до управління вимогами, розглянуто методи їх виявлення, класифікації, аналізу та валідації, досліджено особливості застосування спеціалізованих інструментів для підтримки процесу управління вимогами та проведено оцінку їх ефективності.

У першому розділі розглянуто теоретичні основи управління вимогами до програмного забезпечення, визначено основні види вимог та досліджено сучасні інструменти управління вимогами.

У другому розділі проаналізовано методи виявлення та категоризації вимог, процеси їх розробки, документування, аналізу та валідації.

У третьому розділі досліджено структуру процесу управління вимогами, виконано оцінку сучасних інструментів управління вимогами та проведено аналіз результатів їх практичного застосування.

Висновок: ефективне управління вимогами є одним із ключових факторів успішної розробки програмного забезпечення, оскільки дозволяє підвищити якість кінцевого продукту, знизити ризики виникнення помилок та забезпечити відповідність програмної системи потребам користувачів.

КЛЮЧОВІ СЛОВА: УПРАВЛІННЯ ВИМОГАМИ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ЖИТТЄВИЙ ЦИКЛ ПЗ, АНАЛІЗ ВИМОГ, ВАЛІДАЦІЯ ВИМОГ, ДОКУМЕНТУВАННЯ ВИМОГ, ІНСТРУМЕНТИ УПРАВЛІННЯ ВИМОГАМИ, РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.

ANNOTATION

The bachelor's thesis contains of 78 pages, 6 figures, 10 tables, and a reference list containing 40 sources.

The aim of the work is to study software requirements management methods, analyze modern approaches to requirements elicitation.

Research object: the process of software requirements management.

Research subject: methods, models, and tools for managing software requirements throughout the software development life cycle.

Research results: modern approaches to requirements management were analyzed; methods of requirements elicitation, classification, analysis, and validation were investigated; the application of specialized requirements management tools was examined, and their effectiveness was evaluated.

The first section presents the theoretical foundations of software requirements management, identifies the main types of requirements, and reviews modern requirements management tools.

The second section analyzes methods of requirements elicitation and categorization, as well as processes of requirements development, documentation, analysis, and validation.

The third section examines the structure of the requirements management process, evaluates modern requirements management tools, and discusses the results of their practical application.

Conclusion: effective requirements management is one of the key factors in successful software development, as it improves software quality, reduces the risk of defects, and ensures compliance with stakeholder and user needs.

KEY WORDS: REQUIREMENTS MANAGEMENT, SOFTWARE ENGINEERING, SOFTWARE DEVELOPMENT LIFE CYCLE, REQUIREMENTS ANALYSIS, REQUIREMENTS VALIDATION, REQUIREMENTS DOCUMENTATION, REQUIREMENTS MANAGEMENT TOOLS, SOFTWARE DEVELOPMENT.

ЗМІСТ

ВСТУП

У сучасному світі програмне забезпечення є невід'ємною складовою більшості інформаційних систем, бізнес-процесів та цифрових сервісів. Зростання складності програмних продуктів, збільшення кількості зацікавлених сторін та постійні зміни бізнес-потреб підкреслюють важливість ефективного управління вимогами до програмного забезпечення. Неповні, нечіткі або суперечливі вимоги часто стають причиною затримок у розробці, перевищення бюджету та зниження якості кінцевого продукту. Саме тому процес управління вимогами відіграє ключову роль у забезпеченні успішної реалізації програмних проєктів.

Актуальність теми зумовлена необхідністю застосування сучасних методів управління вимогами, які забезпечують їх своєчасне виявлення, аналіз, документування, відстеження та контроль змін протягом усього життєвого циклу програмного забезпечення. В умовах швидкого розвитку інформаційних технологій та поширення гнучких методологій розробки особливого значення набувають інструменти та підходи, що дозволяють ефективно координувати взаємодію між замовниками, аналітиками, розробниками та тестувальниками.

Метою роботи є дослідження методів управління вимогами до програмного забезпечення, аналіз сучасних підходів та інструментів, а також оцінка їх ефективності у процесі розробки програмних систем.

Для досягнення поставленої мети необхідно виконати **такі завдання**: дослідити теоретичні основи управління вимогами до програмного забезпечення, розглянути підходи до визначення та класифікації вимог, проаналізувати сучасні інструменти управління вимогами, дослідити процеси виявлення, документування та аналізу вимог, оцінити ефективність використання різних методів та засобів управління вимогами, а також сформулювати рекомендації щодо їх практичного застосування.

Об'єктом дослідження є процес управління вимогами до програмного забезпечення на різних етапах його життєвого циклу.

Предметом дослідження є методи, моделі, підходи та інструменти, що використовуються для збору, аналізу, документування, відстеження та управління змінами вимог у програмних проєктах.

Методи дослідження включають аналіз наукової та технічної літератури, порівняльний аналіз існуючих підходів і засобів управління вимогами, систематизацію отриманих результатів, а також оцінювання ефективності сучасних інструментів підтримки процесу управління вимогами.

Практичне значення роботи полягає у можливості використання отриманих результатів для підвищення якості процесів розробки програмного забезпечення, зниження ризиків виникнення помилок на ранніх етапах проєкту та покращення взаємодії між учасниками процесу розробки.

Бакалаврська робота містить 78 сторінок, 6 рисунків, 10 таблиць, 3 розділи, список використаних джерел із 40 найменувань.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ УПРАВЛІННЯ ВИМОГАМИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Основи управління вимогами до програмного забезпечення

Управління вимогами – це організований процес документування, відстеження, управління та визначення пріоритетів розрізнених частин вимог. Це безперервний процес, який проходить протягом усієї розробки програмного забезпечення та вимагає участі всіх зацікавлених сторін для досягнення згоди щодо потенційних змін у вимогах [1]. Через діяльність, пов'язану з управлінням вимогами, це один з найбільш залежних від співпраці процесів у розробці програмного забезпечення. Фактори, які є вирішальними для успішного управління вимогами, включають: комунікацію та координацію між зацікавленими сторонами; створення та управління спільними репозиторіями для вимог; відстеження вимог; та управління змінами вимог.

Глобальна розробка програмного забезпечення (GSD), де команди розробників програмного забезпечення розташовані в різних частинах світу, стає все більш популярною [2]. Щоб отримати переваги від зниження вартості розробки програмного забезпечення та доступу до міжнародних талантів, GSD є гарним рішенням для багатьох організацій. Кількість організацій, що займаються GSD, зростає, і в це вкладено значно більше коштів [3]. Незважаючи на переваги, GSD запровадив проблеми для зацікавлених сторін, яких немає в проектах програмного забезпечення, розроблених в одному місці (колокаційні проекти). Через те, що команди розробників знаходяться в різних географічних місцях, відмінності в культурах та часових поясах негативно впливають на процеси комунікації та координації. Як наслідок, частота комунікації, координації та довіри між командами розробників зменшується [4]. Крім того, різні підходи та процеси розробки програмного забезпечення, технічні можливості віддалено розподілених членів команди та менша видимість робіт з розробки, що виконуються на різних місцях, створюють додаткові проблеми для зацікавлених

сторін. З основних проблем, з якими стикається успішна GSD, найбільше значення має управління вимогами.

Наявність культурних, мовних, соціальних, часових та географічних відмінностей не лише впливає на процес комунікації в GSD, але й створює труднощі для команд розробників у створенні та управлінні спільними репозиторіями вимог. Ці репозиторії є спільним місцем для різних команд розробників, де вони записують та обмінюються деталями проекту з іншими командами. Хоча створення та управління цими репозиторіями здається простим процесом, через різні процеси та стандарти розробки програмного забезпечення дані, записані однією командою, часто стають несумісними з даними, записаними іншими командами, і тому не можуть бути ефективно використані для відстеження вимог та подальших процесів розробки. Аналогічно, для управління змінами для забезпечення дотримання вимог, окрім процесу управління змінами вимог (RCM), потрібна комунікація та координація між зацікавленими сторонами, чого важко досягти в GSD.

Управління вимогами є складним та проблематичним у середовищі GSD. Вищезазначені проблеми не вирішуються належним чином існуючими методами управління вимогами [5]. У цій роботі ми представляємо метод управління вимогами на основі онтологій для проектів GSD. Онтологія в програмній інженерії використовується для визначення вимог у документі специфікації вимог до програмного забезпечення (SRS), дослідження проектування системи, організації програмної системи в підсистеми, зберігання інформації в репозиторії вимог та оцінки вимог на предмет невідповідностей, а також для сприяння отриманню та пошуку знань, обміну вимогами та обслуговування системи [6]. Наш метод допомагає зацікавленим сторонам виконувати такі дії:

(1) створити та підтримувати репозиторій вимог;

(2) створити матрицю відстеження вимог для встановлення двосторонніх зв'язків між артефактами, що містять вимоги, та артефактами, що не є вимогами;

(3) обмінюватися та обговорювати знання про проект між командами розробників;

(4) керувати змінами у вимогах.

Попередні дослідники використовували студентські групи в університетському середовищі для виконання ролей зацікавлених сторін в експериментах у дослідженнях GSD [6]. Так само ми перевіряємо наш метод, застосовуючи його до тематичного дослідження системи онлайн-магазинів, де ролі клієнта, інженера з вимог, аналітика проекту та розробників виконувала група студентів. Протягом усієї роботи ми називаємо ці групи людей «зацікавленими сторонами».

Вимоги до програмного забезпечення виражають потреби та обмеження, що накладаються на програмний продукт, які сприяють задоволенню потреб певного реального застосування [7]. Застосування може, наприклад, вирішувати певну бізнес-проблему або використовувати бізнес-можливості, що пропонується новим ринком. Важливо розуміти, що, за винятком випадків, коли проблема мотивована технологією, проблема є артефактом проблемної області та, як правило, є технологічно нейтральною. Програмний продукт сам по собі може задовольнити цю потребу (наприклад, якщо це настільний додаток), або він може бути компонентом (наприклад, модулем стиснення мовлення, що використовується в мобільному телефоні) програмно-місткої системи, для якої задоволення потреби є емерджентною властивістю. Фундаментально кажучи, спосіб обробки вимог для окремих продуктів та компонентів програмно-містких систем є однаковим.

Однією з головних цілей інженерії вимог є виявлення способів розділення системи; визначення того, які вимоги слід розподілити між якими компонентами. У деяких системах усі компоненти будуть реалізовані програмно. Інші складатимуться з комбінації технологій. Майже всі вони матимуть користувачів-людей, і іноді має сенс враховувати всі компоненти системи, яким слід розподілити вимоги (наприклад, для економії коштів або використання людської адаптивності та винахідливості). Через це інженерія вимог є фундаментальною діяльністю системної інженерії, а не специфічною для програмної інженерії. У цьому відношенні термін «інженерія вимог до програмного забезпечення» є

оманливим, оскільки він передбачає вузьку сферу застосування, пов'язану лише з обробкою вимог, які вже були отримані та розподілені між компонентами програмного забезпечення. Оскільки практикуючі інженери-програмісти все частіше беруть участь у виявленні та розподілі вимог, важливо, щоб сфера знань поширювалася на весь процес інженерії вимог.

Одним із фундаментальних принципів якісної розробки програмного забезпечення є наявність гарної комунікації між користувачами системи та розробниками систем. Саме інженер-технічний спеціаліст є посередником для цієї комунікації. Він повинен бути посередником між сферою діяльності користувача системи (та інших зацікавлених сторін) та технічним світом інженера-програміста. Це вимагає від нього технічних навичок, здатності розуміти сферу застосування та міжособистісних навичок, щоб допомогти досягти консенсусу між різнорідними групами зацікавлених сторін.

Важливість управління вимогами (RM) в інженерії вимог зараз визнана. Це важливий засіб забезпечення того, щоб діяльність з розробки програмного забезпечення базувалася на еталонних вимогах, тим самим забезпечуючи якість системи. RM складається з підтримки базової лінії вимог (RBM), відстеження вимог (RT) та управління змінами вимог (RCM) [8].

Машинне навчання (МН) – це міждисциплінарна область, що охоплює теорію ймовірностей, статистику, теорію складності алгоритмів тощо. Її можна використовувати для моделювання поведінки людини під час навчання, отримання нових знань чи навичок та реорганізації існуючих структур знань для постійного покращення їхньої продуктивності [9]. Дослідження МН йдуть повним ходом і широко використовуються в експертних системах, обробці природної мови, розпізнаванні образів, комп'ютерному зорі та інших галузях.

У RM багато методів машинного навчання було використано для вирішення цих проблем на основі середовища програмно-визначеного Інтернету речей. З точки зору RBM, поточні дослідження в основному зосереджені на тому, як покращити якість та зрозумілість специфікацій вимог. Наприклад, використовують алгоритм класифікації для вирішення проблеми невідповідності

та неповноти великих специфікацій вимог шляхом автоматичної класифікації вимог. Поєднали граматичний аналіз та методи навчання з учителем, щоб розділити вимоги для покращення зрозумілості у специфікації вимог.

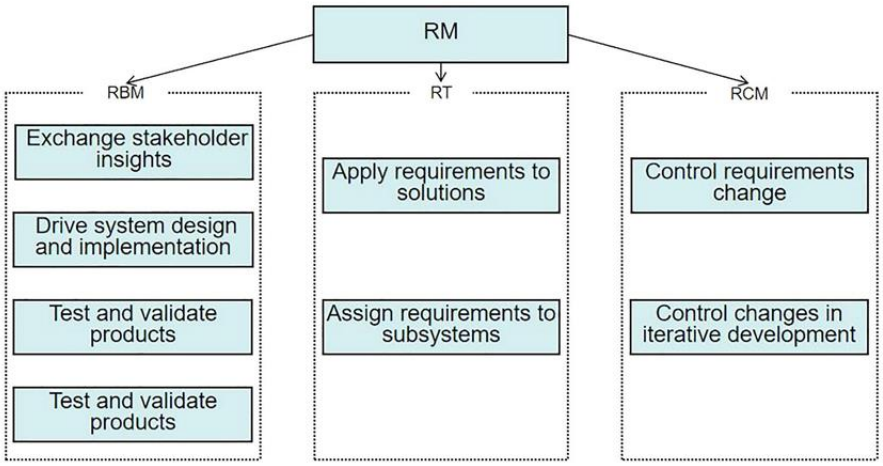

Рисунок 1.1 - Діяльності та їхні основні завдання в рамках управління вимогами.

Що стосується RT, деякі дослідники поєднують метод машинного навчання з традиційними методами відстеження для підвищення ефективності та результативності, такими як поєднання структурного аналізу або кластеризації з методами пошуку інформації для підвищення точності та швидкості відтворення посилань RT [10]. Деякі люди поєднують методи класифікації з методами обробки природної мови для прогнозування кількості позитивних посилань.

Третя частина RM – це RCM, і зміни неминучі зі зміною середовища. Метод машинного навчання RCM в основному включає дві категорії: метод ML для контролю поширення змін попиту та метод ML для прогнозування змін попиту. Для першої категорії використовували інтерактивний генетичний алгоритм для контролю поширення змін вимог, які можуть відбуватися з часом. Для другої категорії використовували підхід логістичної регресії для прогнозування можливих змін вимог у вбудованих системах; використовували підхід багатоміткового навчання для прогнозування можливого поширення змін

вимог; використовували модель випадкового лісу для прогнозування можливих змін вимог під час ітераційного процесу .

RM (Управління управлінням ресурсами) може забезпечити прозорість процесу розробки програмного забезпечення, реалізувати управління змінами вимог та прогнозування для програмного забезпечення, що є гарантією успіху програмного забезпечення. ML може покращити продуктивність шляхом реорганізації існуючих структур знань на основі даних або попереднього досвіду. Це надає нові ідеї та рішення для підтримки великої кількості специфікацій вимог, відстеження вимог у різних продуктах та управління динамічними вимогами в RM. Різні методи ML, згадані в роботі, були використані для вирішення проблем, що виникають в RM. Однак, комплексний огляд методів ML, що використовуються в RM, був відсутній. Тому в цій роботі використовується дослідження зіставлення для систематичного аналізу методів ML для RM. Це зіставлення визначає фактори, що впливають на методи ML для RM, визначає цілі методів ML для RM, підсумовує методи ML, що використовуються в RM, та аналізує метрики оцінки методів ML, що використовуються в RM. Ми наполягаємо на тому, що огляд був корисним як для дослідників, так і для практиків у галузі управління ресурсами, оскільки він надає розуміння, яке може допомогти дослідникам і практикам отримати добре розуміння методів машинного навчання для управління ресурсами. Дослідники, зацікавлені в цій галузі, можуть використовувати цю роботу як карту для зручного пошуку відповідних досліджень та виявлення тих областей, які потребують подальших досліджень. Для практиків це зіставлення дає детальне порівняння ефективності багатьох методів машинного навчання, які використовувалися в управлінні ресурсами, що є орієнтиром для них у виборі відповідного методу машинного навчання на практиці для вирішення конкретної проблеми управління ресурсами [11].

-1.2 Визначення та класифікація вимог до програмного забезпечення

У цьому підрозділі наведено огляд інженерії вимог, в якій:

- визначено поняття «вимога»;
- визначаються мотивації для систем та обговорюється їхній зв'язок з вимогами;
- описано загальний процес аналізу вимог, а потім обговорено, чому на практиці організації часто відхиляються від цього процесу; та
- Описано результати процесу розробки вимог та необхідність управління вимогами.

Цей огляд має на меті надати перспективу або «точку зору» на галузь знань, яка доповнює огляд – Розподіл тем для галузі знань «Вимоги до програмного забезпечення».

У своїй найпростішій формі вимога – це властивість, яку необхідно продемонструвати для вирішення певної проблеми реального світу [12]. У цьому документі йдеться про вимоги до «систем», а не до «рішень», оскільки він стосується проблем, які мають програмні рішення. Отже, вимога – це властивість, яку необхідно продемонструвати системі, розробленій або адаптованій для вирішення конкретної проблеми. Проблема може полягати в автоматизації частини завдання того, хто використовуватиме систему, у підтримці бізнес-процесів організації, яка замовила систему, у виправленні недоліків існуючої системи, в управлінні пристроєм та багато іншого. Функціонування користувачів, бізнес-процесів та пристроїв зазвичай є складним. Отже, в ширшому сенсі вимоги до системи зазвичай є складною комбінацією вимог від різних людей на різних рівнях організації та від середовища, в якому система повинна працювати.

Вимоги різняться за призначенням та типами властивостей, які вони представляють. Можна розрізнити *параметри продукту* та *параметри процесу.* Параметри продукту – це вимоги до системи, що розробляється, і їх можна додатково класифікувати як [13]:

- Функціональні вимоги до системи, такі як форматування тексту або модуляція сигналу. Функціональні вимоги іноді називають можливостями.

♦ Нефункціональні вимоги, що обмежують рішення. Нефункціональні вимоги іноді

відомі як обмеження або вимоги до якості. Їх можна додатково класифікувати залежно від того, чи є вони (наприклад) вимогами до продуктивності, вимогами до ремонтопридатності, вимогами безпеки, вимогами до надійності, електромагнітними вимоги сумісності та багато інших типів вимог.

Параметр процесу, по суті, є обмеженням на розробку системи (наприклад, «програмне забезпечення має бути написано на Ada»). Іноді його називають вимогами до процесу.

Вимоги мають бути чітко та однозначно сформульовані, а де це доречно, кількісно. Важливо уникати розпливчастих та неперевірених вимог, інтерпретація яких залежить від суб'єктивної оцінки («система має бути надійною», «система має бути зручною для користувача»). Це особливо важливо для нефункціональних вимог. Два приклади кількісних вимог: система повинна збільшити пропускну здатність кол-центру на 20%; та вимога, що система повинна мати ймовірність виникнення фатальної помилки протягом будь-якої години роботи менше $1 * 10^{-6}$. Вимога до пропускної здатності є дуже високою і її потрібно буде використовувати для виведення низки детальних вимог. Вимога надійності жорстко обмежуватиме архітектуру системи [14].

Деякі вимоги є емерджентними властивостями. Тобто вимогами, які не можуть бути виконані одним компонентом, але задоволення яких залежить від того, як взаємодіють усі компоненти системи. Вимога до пропускної здатності кол-центру, наведена вище, наприклад, залежатиме від того, як телефонна система, інформаційна система та оператори взаємодіяли б у реальних умовах експлуатації. Емерджентні властивості вирішально залежать від архітектури системи.

Істотною властивістю всіх вимог є їхня перевіреність. Перевірка певних вимог може бути складною або дорогою. Наприклад, перевірка вимог до пропускної здатності кол-центру може вимагати розробки програмного

забезпечення для моделювання. Персонал з розробки вимог та перевірки та верифікації повинен забезпечити, щоб вимоги можна було перевірити в межах обмежень доступних ресурсів.

Деякі вимоги породжують неявні вимоги до процесу. Вибір методу перевірки є одним із прикладів. Іншим може бути використання особливо суворих методів аналізу (таких як методи формальних специфікацій) для зменшення системних помилок, які можуть призвести до недостатньої надійності. Вимоги до процесу також можуть бути встановлені безпосередньо організацією-розробником, її замовником або третьою стороною, такою як регулятор безпеки.

Вимоги мають інші атрибути, окрім поведінкової властивості, яку вони виражають. Типовими прикладами є рейтинг пріоритету, який дозволяє йти на компроміси за умови обмежених ресурсів, та значення статусу, яке дозволяє контролювати хід виконання проекту. Кожна вимога повинна бути унікально ідентифікована, щоб її можна було контролювати конфігурацією та керувати нею протягом усього життєвого циклу системи.

Системні вимоги та драйвери процесів

У літературі з інженерії вимог системні вимоги іноді називають «вимогами користувача». Ми надаємо перевагу вужчому визначенню терміна «вимоги користувача», в якому вони позначають вимоги людей, які будуть клієнтами системи або кінцевими користувачами. Системні вимоги, навпаки, включають вимоги користувачів, вимоги інших зацікавлених сторін (таких як регуляторні органи) та вимоги, які не мають ідентифікованого людського джерела. Типовими прикладами зацікавлених сторін системи є (але не обмежуються):

- Користувачі – люди, які керуватимуть системою. Користувачі часто є гетерогенною групою, що складається з людей з різними ролями та вимогами.
- Клієнти – люди, які замовили систему або які представляють цільовий ринок системи.
- Аналітики ринку – продукт масового ринку не матиме замовника-

замовника, тому часто потрібні маркетологи, щоб визначити потреби ринку та діяти як посередники-клієнти.

- Регулятори – багато прикладних сфер, таких як банківська справа та громадський транспорт, регулюються. Системи в цих сферах повинні відповідати вимогам регуляторних органів.
- Розробники систем – вони мають законний інтерес в отриманні прибутку від розробки системи, наприклад, шляхом повторного використання компонентів у різних продуктах. Якщо в цьому випадку клієнт певного продукту має конкретні вимоги, які обмежують можливість повторного використання компонентів, розробник повинен ретельно зважити власні інтереси з інтересами замовника. Для продуктів масового ринку розробник часто є основною зацікавленою стороною, оскільки він хоче підтримувати продукт на якомога більшому ринку якомога довше.

Окрім цих людських джерел вимог, важливі системні вимоги часто походять від інших пристроїв або систем у середовищі, які вимагають певних послуг системи або діють для обмеження системи, або навіть від фундаментальних характеристик області застосування [15]. Наприклад, бізнес-система може бути зобов'язана взаємодіяти зі старою базою даних, а багато військових систем повинні бути стійкими до високих рівнів електромагнітного випромінювання. Ми говоримо про «виявлення» вимог, але на практиці інженер з вимог повинен систематично витягувати та інвентаризувати вимоги з комбінації зацікавлених сторін-людей, середовища системи, техніко-економічних обґрунтувань, аналізу ринку, бізнес-планів, аналізу конкуруючих продуктів та знань предметної області [16].

Визначення та аналіз системних вимог повинні ґрунтуватися на необхідності досягнення загальних цілей проекту. Для забезпечення цієї спрямованості слід скласти бізнес-кейс, який чітко визначає переваги, які повинні принести інвестиції. Це має слугувати «перевіркою реальності», яку можна застосувати до системних вимог, щоб гарантувати, що фокус проекту не зміщується. Якщо є сумніви щодо технічної, операційної чи фінансової

життєздатності проекту, слід провести аналіз доцільності. Він розроблений для виявлення ризиків проекту та оцінки ступеня, до якого вони загрожують життєздатності системи. Ризики повинні бути задокументовані в плані управління проектом.

Типові ризики включають здатність задовольнити нефункціональні вимоги, такі як продуктивність або наявність готових компонентів. У деяких спеціалізованих галузях може знадобитися розробити симуляції для отримання даних, що дозволять оцінити ризики проекту. У таких галузях, як громадський транспорт, де безпека є проблемою, слід провести аналіз небезпек, на основі якого можна визначити вимоги безпеки.

Огляд аналізу вимог

Після визначення цілей проекту можна розпочати роботу з виявлення, аналізу та перевірки вимог до системи. Це має вирішальне значення для отримання чіткого розуміння проблеми, яку система має вирішити, та її ймовірної вартості [17].

Інженер з вимог повинен прагнути повноти, забезпечуючи визначення та консультації з усіма відповідними джерелами вимог. Зазвичай консультації з усіма бувають неможливими. Наприклад, може бути багато користувачів великої системи. Однак слід визначити та проконсультуватися з репрезентативними прикладами кожного класу зацікавлених сторін системи. Хоча окремі зацікавлені сторони будуть авторитетними щодо аспектів системи, які представляють їхні інтереси або досвід, інженер з вимог несе відповідальність за створення «загальної картини», щоб забезпечити повноту з усіма окремими зацікавленими сторонами.

Виявлення вимог зацікавлених сторін рідко буває легким, і інженер з вимог повинен вивчити низку методів, які допомагають людям чітко сформулювати, як вони виконують свою роботу та що допоможе їм краще виконувати свою роботу. Існує багато соціальних та політичних питань, які можуть впливати на вимоги зацікавлених сторін та їхню здатність чи бажання їх формулювати, і необхідно бути чутливим до них [18]. У багатьох випадках

необхідно забезпечити контекстуальну основу, яка служить для фокусування консультацій; щоб допомогти зацікавленій стороні визначити, що можливо, та допомогти інженеру з вимог перевірити їхнє розуміння. Ознайомлення зацікавлених сторін з прототипами може допомогти, і вони не обов'язково мають бути високої точності. Серія ескізів на фліпчарті іноді може служити тій самій меті, що й прототип програмного забезпечення, уникаючи при цьому пасток відволікання уваги, спричинених косметичними особливостями програмного забезпечення. Проведення зацікавленої сторони через невелику кількість сценаріїв, що представляють послідовності подій у сфері застосування, також може допомогти зацікавленій стороні та інженеру з вимог дослідити ключові фактори, що впливають на вимоги.

Після визначення, системні вимоги повинні бути перевірені зацікавленими сторонами та узгоджені компроміси, перш ніж будуть виділені додаткові ресурси на проект. Для забезпечення валідації системні вимоги зазвичай тримаються на високому рівні та виражаються з точки зору області застосування, а не в технічних термінах. Отже, системні вимоги для інтернет-книгарні будуть виражені з точки зору книг, авторів, складських приміщень та транзакцій за кредитними картками, а не з точки зору протоколів зв'язку чи алгоритмів розподілу ключів, які можуть бути частиною рішення. Занадто багато технічних деталей на цьому етапі затьмарює основні характеристики системи з точки зору її клієнта та користувачів.

Деякі системні вимоги можуть бути невиконавними. Деякі можуть бути технічно нездійсненними, інші можуть бути занадто дорогими для реалізації, а деякі будуть взаємосумісними. Інженер з вимог повинен проаналізувати вимоги, щоб зрозуміти їхні наслідки та взаємодію. Їх необхідно визначити за пріоритетами та оцінити їх вартість. Мета полягає у визначенні обсягу системи та «базового» набору системних вимог, який є здійсненним та прийнятним. Це може вимагати допомоги зацікавленим сторонам, чиї вимоги суперечать (одно одному або через витрати чи інші обмеження), у домовленості про прийнятні компроміси.

Для полегшення аналізу системних вимог будуються концептуальні моделі системи. Вони допомагають зрозуміти логічний поділ системи, її контекст в операційному середовищі, а також зв'язок даних і керування між логічними сутностями. Загалом, для дослідження різних аспектів системи та її проблемної області слід розробити поєднання статичних (наприклад, об'єктна модель) та динамічних (наприклад, трасування подій та діаграми станів) моделей. Однак вибір аспектів для моделювання залежить від характеру проблемної області.

Системні вимоги необхідно аналізувати в контексті всіх застосовних обмежень. Обмеження походять з багатьох джерел, таких як бізнес-середовище, організаційна структура замовника та операційне середовище системи. Вони включають бюджет, графік, технічні (нефункціональні вимоги), регуляторні та інші обмеження. Отже, робота інженера з вимог не обмежується виявленням вимог зацікавлених сторін, а включає оцінку їхньої доцільності. Вимоги, які є явно нездійсненними, слід відхилити, а причину відхилення зафіксувати. Вимоги, щодо яких існує лише підозра у нездійсненності, є складнішими. Дослідження доцільності може бути виправданим, якщо, наприклад, зацікавлені сторони рішуче підтримують сумнівну вимогу [19].

Ресурси проекту повинні бути зосереджені на найважливіших пріоритетних вимогах. В принципі, вимоги повинні бути одночасно *необхідними та достатніми* – не повинно бути нічого пропущеного або чогось, що не потрібно включати. Досягти цього, звичайно, складно. Відсутність важливої інформації про вимоги можна виявити лише шляхом ретельного аналізу. Так само може знадобитися значний зусилля для досягнення консенсусу щодо пріоритетів вимог, оскільки суттєва вимога однієї зацікавленої сторони може мати лише косметичну цінність для іншої. На практиці наявність достатніх ресурсів дозволить задовольнити деякі несуттєві вимоги, тоді як недостатня кількість ресурсів може змусити виключити навіть наполегливо пропаговані вимоги. Незалежно від того, як визначено базову лінію, персонал з питань вимог та верифікації повинен провести приймальні тести, які

гарантуватимуть відповідність вимогам перед постачанням або випуском продукту.

Зрештою, в результаті процесу аналізу з'явиться повний та узгоджений набір системних вимог. На цьому етапі основні сфери функціональності мають бути чіткими. Визначаються підсистеми або компоненти для обробки кожної основної сфери функціональності. Потім системні вимоги розподіляються між підсистемами/компонентами.

Ця діяльність з розділення та розподілу є частиною архітектурного проектування. Архітектурне проектування – це навичка, яка залежить від багатьох факторів, таких як розпізнавання багаторазових архітектурних «шаблонів» або існування готових компонентів. Розробка архітектури системи є важливою віхою в проекті, і вкрай важливо правильно її опанувати. Зокрема, взаємодія компонентів системи вирішально впливає на те, якою мірою система демонструватиме бажані нові властивості. На цьому етапі вимоги до системи та архітектура системи документуються, переглядаються та «підписуються» як базова лінія для подальшої розробки, планування проекту та оцінки вартості.

За винятком маломасштабних систем, розробникам програмного забезпечення зазвичай неможливо розпочати детальне проектування системних компонентів з документа з вимогами до системи. Вимоги, призначені для компонентів, які самі по собі є складними системами, потребуватимуть подальших циклів аналізу, щоб додати більше деталей та інтерпретувати системні вимоги, орієнтовані на предметну область, для розробників, які можуть не мати достатніх знань про предметну область застосування, щоб правильно їх інтерпретувати. Отже, з кожної системної вимоги високого рівня зазвичай виводиться низка детальних технічних вимог. Вкрай важливо записувати та підтримувати цей похідний код, щоб можна було відстежувати вимоги. Відстеження має вирішальне значення для управління вимогами, оскільки воно дозволяє, наприклад, оцінити вплив будь-яких наступних змін до вимог.

Удосконалення вимог та архітектури системи – це те, де інженерія вимог поєднується з проектуванням програмного забезпечення. Чітких меж немає, але

аналіз вимог рідко триває далі 2 або 3 рівнів архітектурної декомпозиції, перш ніж відповідальність за окремі компоненти передається командам проектувальників.

Інженерія вимог на практиці

Огляд аналізу вимог, описував процес виявлення та аналізу вимог і побудови архітектури системи так, ніби це лінійна послідовність дій. Це ідеалізований погляд на процес. У цьому розділі розглядаються деякі причини, чому лінійний процес рідко буває практичним у контексті реальних програмних проектів.

У індустрії програмного забезпечення існує загальний тиск на скорочення циклів розробки, і це особливо помітно у висококонкурентних секторах, що орієнтовані на ринок. Більше того, більшість проектів певним чином обмежені своїм середовищем, і багато з них є оновленнями або переглядом існуючих систем, де архітектура системи є заданою. Тому на практиці майже завжди недоцільно впроваджувати інженерію вимог як лінійний, детермінований процес, де системні вимоги виявляються у зацікавлених сторін, визначаються на основі базових значень, розподіляються та передаються команді розробників програмного забезпечення. Це, безумовно, міф, що вимоги до великих систем коли-небудь ідеально зрозумілі або ідеально визначені [20].

Натомість, вимоги зазвичай ітеративно спрямовані до рівня якості та деталізації, достатнього для прийняття рішень щодо проектування та закупівель. У деяких проектах це може призвести до того, що вимоги будуть визначені до того, як усі їхні властивості будуть повністю зрозумілі. Це ризикує дорогою переробкою, якщо проблеми виникнуть на пізніх етапах процесу розробки. Однак, інженери вимог неминуче обмежені планами управління проектами і тому повинні вживати заходів для забезпечення максимально високої якості вимог з урахуванням наявних ресурсів. Наприклад, вони повинні чітко вказати будь-які припущення, що лежать в основі вимог, та будь-які відомі проблеми.- Навіть там, де інженерія вимог має достатньо ресурсів, рівень аналізу рідко застосовується однаково. Наприклад, на ранній стадії процесу аналізу досвідчені

інженери часто здатні визначити, де існуючі або готові рішення можна адаптувати для впровадження системних компонентів. Вимоги, призначені для них, не потребують подальшої розробки, тоді як інші, для яких рішення менш очевидне, можуть потребувати подальшого аналізу. Критичні вимоги, такі як ті, що стосуються громадської безпеки, завжди повинні бути ретельно проаналізовані.

Майже у всіх випадках розуміння вимог продовжує розвиватися в міру проходження проектування та розробки. Це часто призводить до перегляду вимог на пізніх етапах життєвого циклу. Мабуть, найважливішим моментом розуміння інженерії вимог є те, що значна частина вимог *змінюватиметься* . Іноді це відбувається через помилки в аналізі, але часто є неминучим наслідком змін у «середовищі»: операційному чи бізнес-середовищі клієнта; або на ринку, на якому система повинна продавати, наприклад. Якою б не була причина, важливо визнати неминучість змін та вжити заходів для пом'якшення наслідків змін. Змінами необхідно керувати, забезпечуючи проходження запропонованих змін через визначений процес перегляду та затвердження, а також застосовуючи ретельне відстеження вимог, аналіз впливу та управління версіями. Отже, процес інженерії вимог — це не просто завдання на початку розробки програмного забезпечення, а охоплює весь життєвий цикл розробки. У типовому проекті діяльність інженера з вимог з часом розвивається від виявлення до управління змінами.

Продукти та результати

Хороша інженерія вимог вимагає визначення продуктів процесу – кінцевих результатів (поставок). Найфундаментальнішим з них в інженерії вимог є документ вимог. Він часто складається з двох окремих документів (на цьому етапі також може бути розроблений опис архітектури – див. опис галузі знань для проектування програмного забезпечення):

Документ, що визначає системні вимоги. Іноді його називають документом визначення вимог, документом вимог користувача або, як визначено в стандарті IEEE std 1362-1998, документом концепції операцій (ConOps). Цей

документ служить для визначення високорівневих системних вимог з точки зору зацікавлених сторін. Він також служить засобом для перевірки системних вимог. Його читацька аудиторія включає представників зацікавлених сторін системи. Тому він має бути сформульований з точки зору предметної області замовника. Окрім переліку системних вимог, визначення вимог повинно містити довідкову інформацію, таку як формулювання загальних цілей системи, опис її цільового середовища та формулювання обмежень і нефункціональних вимог до системи. Воно може включати концептуальні моделі, розроблені для ілюстрації системного контексту, сценаріїв використання, основних сутностей предметної області, а також даних, інформації та робочих процесів [21].

Документ, що визначає вимоги до програмного забезпечення. Іноді його називають специфікацією вимог до програмного забезпечення (SRS). Мета та читацька аудиторія SRS дещо відрізняються від документа з визначенням вимог. Грубо кажучи, SRS документує детальні вимоги, отримані з системних вимог і які були призначені для програмного забезпечення. Нефункціональні вимоги у визначенні вимог повинні бути детально розроблені та кількісно визначені. Можна припустити, що основна читацька аудиторія SRS має певні знання з концепцій програмної інженерії. Це може відображатися в мові та нотаціях, що використовуються для опису вимог, а також у деталізації моделей, що використовуються для ілюстрації системи. Для програмного забезпечення на замовлення SRS може бути основою договору між розробником і замовником [22].

Документи з вимогами повинні бути структуровані таким чином, щоб мінімізувати зусилля, необхідні для читання та пошуку інформації в них. Невиконання цієї вимоги знижує ймовірність того, що система відповідатиме вимогам. Це також перешкоджає можливості вносити контрольовані зміни до документа, оскільки система та її вимоги розвиваються з часом. Такі стандарти, як IEEE std 1362-1998 та IEEE std 830-1998, надають шаблони для документів з вимогами. Такі стандарти призначені для загального використання та потребують адаптації до контексту, в якому вони використовуються.

Також слід подбати про те, щоб описувати вимоги якомога точніше. Вимоги зазвичай пишуться природною мовою, але в SRS це може бути доповнено формальними або напівформальними описами. Вибір відповідних позначень дозволяє описати конкретні вимоги та аспекти архітектури системи точніше та лаконічніше, ніж природною мовою. Загальне правило полягає в тому, що слід використовувати позначення, які дозволяють описати вимоги якомога точніше. Це особливо важливо для систем, критично важливих для безпеки, та деяких інших типів надійних систем. Однак вибір позначень часто обмежується підготовкою, навичками та уподобаннями авторів та читачів документа.

Природна мова має багато серйозних недоліків як засіб опису. Серед найсерйозніших є те, що вона неоднозначна та важко точно описати складні поняття. Формальні позначення, такі як Z або CSP, уникають проблеми неоднозначності, оскільки їхній синтаксис та семантика формально визначені. Однак такі позначення недостатньо виразні, щоб адекватно описати кожен аспект системи. Природна мова, навпаки, надзвичайно багата та здатна описати, хоч і недосконало, майже будь-яке поняття чи властивість системи. Природна мова також, ймовірно, є єдиною *лінгва франка автора документа та читачів.* Оскільки природної мови не уникнути, інженери вимог повинні бути навчені використовувати мову просто, лаконічно та уникати поширених причин помилкового тлумачення. До них належать:

- довгі речення зі складними підрядними реченнями;
- використання термінів з більш ніж одним правдоподібним тлумаченням (неоднозначність);
- представлення кількох вимог як однієї вимоги;
- непослідовність у використанні термінів, таких як використання синонімів.

Щоб протидіяти цим проблемам, описи вимог часто мають стилізовану форму та використовують обмежену підмножину природної мови. Наприклад, гарною практикою є стандартизація невеликого набору модальних дієслів для

позначення відносних пріоритетів [23]. Наприклад, «shall» зазвичай використовується для позначення обов'язкової вимоги, а «should» – для позначення лише бажаної вимоги. Отже, вимога «Аварійні гальма повинні бути застосовані для зупинки поїзда, якщо ніс поїзда проїжджає повз сигнал НЕБЕЗПЕКА» є обов'язковою.

Документ(и) з вимогами повинні пройти процедури валідації та верифікації. Вимоги повинні бути валідовані, щоб переконатися, що інженер з вимог зрозумів їх. Також важливо перевірити, чи відповідає документ з вимогами стандартам компанії, чи є він зрозумілим, послідовним та повним. Формальні нотації мають важливу перевагу, оскільки вони дозволяють довести дві останні властивості (принаймні, в обмеженому сенсі). Документ(и) повинні бути піддані перевірці різними зацікавленими сторонами, включаючи представників замовника та розробника. Найважливіше, що документи з вимогами повинні бути поміщені під той самий режим управління конфігурацією, що й інші результати процесу розробки [24].

Документ(и) з вимогами є лише найпомітнішим проявом вимог. Вони не містять інформації, яка не є необхідною для читачів документа. Однак ця інша інформація необхідна для управління ними. Зокрема, важливо відстежувати вимоги.

Один із методів відстеження вимог полягає в побудові орієнтованого ациклічного графа (DAG), який записує походження вимог і забезпечує журнали аудиту вимог. Як мінімум, вимоги повинні бути простежені назад до їх джерела (наприклад, від вимоги до програмного забезпечення до вимоги(й) системи, з якої(их) вона була розроблена) та вперед до артефактів проектування або впровадження, які їх реалізують (наприклад, від вимоги до програмного забезпечення до проектного документа для компонента, який її реалізує). Трасування дозволяє керувати вимогами. Зокрема, воно дозволяє проводити аналіз впливу запропонованої зміни до однієї з вимог.

Сучасні інструменти управління вимогами допомагають підтримувати інформацію про відстеження. Зазвичай вони складаються з бази даних вимог та

графічного інтерфейсу користувача:

- зберігати описи та атрибути вимог;
- щоб дозволити автоматичну генерацію DAG трасування;
- дозволити графічне відображення поширення змін вимог;
- генерувати звіти про стан вимог (наприклад, чи були вони проаналізовані, затверджені, впроваджені тощо);
- створювати документи з вимогами, що відповідають обраним стандартам;
- та застосовувати управління конфігурацією до вимог.

Слід зазначити, що не кожна організація має культуру документування та управління вимогами. Динамічні стартапи, які керуються сильним «баченням продукту» та обмеженими ресурсами, часто розглядають документацію вимог як непотрібні накладні витрати. Однак, неминуче, оскільки ці компанії розширюються, їхня клієнтська база зростає, а їхній продукт починає розвиватися, вони виявляють, що їм потрібно відновити вимоги, які мотивували функції продукту, щоб оцінити вплив запропонованих змін. Отже, документація та управління вимогами є основоположними для будь-якого процесу розробки вимог.

1.3 Інструменти управління вимогами до програмного забезпечення

Інструменти управління вимогами допомагають досягти цілей систематичного управління вимогами:

- підтримка отримання, специфікації, групування та атрибуції необроблених вимог
- підтримка їх виведення на більш детальні рівні, збереження та коригування атрибутів
- що дозволяє визначати та пов'язувати тестові випадки та тестові середовища
- що дозволяє відстежувати зв'язки між вимогами, проектуванням,

реалізацією та тестуванням

- забезпечити розподілену розробку в межах організації

Оскільки всі великі проекти розробки систем повинні керувати великими наборами вимог, існує широка потреба в таких інструментах, і ряд з них стали комерційно доступними в останні роки. Ці інструменти використовують різні концепції, мають різні можливості та різний ступінь зрілості щодо їх застосовності в проектах системної інженерії. Оскільки вимоги використовуються та змінюються майже протягом усього процесу розробки, зміна інструментів посеред проекту є дорогою та трудомісткою справою. У випадку модульних лінійок продуктів зміни ще складніше впоратися через численні зв'язки між проектами.

Щоб забезпечити ретельний вибір інструментів, який враховує всі аспекти продукту, у цій роботі описано каталог вимог до інструментів управління ресурсами (RM). Каталог не охоплює загальні аспекти вибору інструментів, такі як ціна, ліцензування, надійність постачальника, стабільність, документація, зручність використання тощо [25]. Натомість він зосереджується на функціональних вимогах до інструментів. Публікуючи набір вимог до інструментів на основі значного досвіду проектів, ми прагнули допомогти стратифікації деяких базових функцій інструменту управління ресурсами та вплинути на майбутній розвиток інструментів.

Вимоги до інструментів управління вимогами дуже різноманітні. Вони залежать від різних факторів, зокрема від предметної області продукту, розміру проекту та організації, обсягу повторного використання та зрілості процесу. Кількість перестановок між цими факторами велика, і жоден інструмент не може охопити всі вимоги. Тим не менш, ми вважаємо, що більшість відмінностей можна покрити, коригуючи пріоритети та використовуючи різні моделі управління вимогами (RMI), які не залежать від інструментів. Отже, одним з наших найважливіших критеріїв є можливість вільного моделювання вимог в інструменті. Невеликим організаціям часто потрібна лише невелика підмножина попередньо визначених атрибутів та типів вимог. Великі організації зацікавлені

у використанні одного й того ж інструменту в багатьох різних проектах, щоб заощадити на витратах на ліцензування, встановлення та підтримку користувачів. Ще одна відмінність полягає в тому, що існують інструменти з сильною підтримкою методів та процесів, а також інструменти з менш суворою підтримкою процесів. Останні більше підходять для команд розробників, які вже мають високий рівень зрілості процесу. Є інструменти з сильною орієнтацією на документи та інші з сильною орієнтацією на бази даних. Перші часто є кращими в системній інженерії, оскільки інженери звикли до специфікацій, подібних до документів. Сильна сторона останніх полягає в тому, що вони підтримують написання високоструктурованих специфікацій.

1.3 Висновок по розділу

У першому розділі було розглянуто теоретичні основи управління вимогами до програмного забезпечення. Визначено сутність процесу управління вимогами, його значення для забезпечення якості програмного продукту та успішної реалізації проєктів. Проаналізовано основні види вимог, їх класифікацію та особливості формування. Також досліджено сучасні інструменти управління вимогами, які забезпечують документування, відстеження змін і підтримку взаємодії між учасниками проєкту. Отримані результати створюють теоретичне підґрунтя для вивчення методів і процесів управління вимогами.

РОЗДІЛ 2. МЕТОДИ ТА ПРОЦЕСИ УПРАВЛІННЯ ВИМОГАМИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Методи виявлення та категоризації вимог

Цей підрозділ стосується категорії методів визначення вимог. Оскільки розробка вимог – це інтенсивний процес взаємодії між зацікавленими сторонами та аналітиками, ми розрізняємо чотири типи методів виявлення вимог відповідно

до засобів комунікації: спостережливий, розмовний, аналітичний та синтетичний. Кожен тип представляє специфічну модель взаємодії між аналітиками та зацікавленими сторонами та відображає природу методу в цьому документі. Розуміння категорії методу допомагає інженерам зрозуміти різні методи виявлення вимог та спрямовує їх у виборі відповідного (набору) методу(ів) для виявлення вимог. Крім того, це надає нам збагачену базу для прогнозування тенденцій у методі виявлення вимог та надання стратегій для розробки вимог.

Розмовні методи

Розмовний метод забезпечує засіб усного спілкування між двома або більше людьми. Оскільки розмова є природним способом вираження потреб та ідей, а також ставити та відповідати на запитання, вона ефективна для розвитку та розуміння проблем і виявлення загальних вимог до продукту. Методи цієї категорії також називають вербальними методами [26]. Типовим розмовним методом є інтерв'ю. Він зазвичай використовується для виявлення вимог [27]. Інші методи цієї категорії включають семінари, фокус-групи, мозковий штурм тощо, як показано в таблиці 2.1.

Розмова є однією з найпоширеніших, але водночас невидимих форм соціальної взаємодії. Люди зазвичай із задоволенням описують свою роботу та труднощі, з якими вони стикаються. Вербально виражені вимоги, потреби та обмеження часто називають неявними вимогами. Оскільки вербальне спілкування є практичним та ефективним для збору неявних знань, розмовні методи формують основний підхід до виявлення неявних вимог. Шляхом проведення інтерв'ю, семінарів або мозкового штурму вимоги формулюються зацікавленими сторонами та повідомляються аналітикам [28].

Таблиця 2.1.

Перелік методів ведення діалогу

Метод (Method)	**Ведучий / Організатор (Conductor)**	**Опис (Description)**
Interviews *(Інтерв'ю)*	Досвідчений аналітик із загальними знаннями про предметну область застосування (An experienced analyst with generic knowledge about the application domain)	Аналітик обговорює бажаний продукт з різними групами людей і формує розуміння їхніх вимог. Якщо інтерв'ю проводиться за заздалегідь визначеним планом і питаннями, воно називається структурованим; в іншому випадку це відкрите інтерв'ю. + Функції продукту (+ Product features)
Workshop, focus groups *(Воркшоп, фокус-групи)*	Досвідчений зовнішній фасилітатор (An experienced outside facilitator)	Представники зацікавлених сторін збираються разом на короткий, але інтенсивно сфокусований період для створення або перегляду високорівневих функцій бажаних продуктів. + Функції продукту (+ Product features)
Brainstorming [19] *(Мозковий штурм)*	Досвідчений зовнішній фасилітатор (An experienced outside facilitator)	Представники зацікавлених сторін збираються разом і швидко формують великий і широкий список ідей. Це заохочує до «нешаблонного» мислення (out-of-the-box) без звичних обмежень і включає як генерацію, так і відбір (скорочення кількості) ідей. + Функції продукту (+ Product features) + Інновації/нові ідеї щодо продуктів (+ Innovation/new ideas regarding products)

Розмовні методи дуже часто використовуються в розробці вимог. Однак вони є трудомісткими [29]: організація зустрічі, створення стенограм та аналіз записів живої взаємодії потребують часу. Водночас, є складним завданням сприяти процесу виявлення потреб, особливо коли використовується семінар або мозковий штурм, наприклад, планування зустрічі та забезпечення присутності представників на зустрічі.

Методи спостереження

Таблиця 2.2.

Перелік методів спостереження

Метод (Method)	Ведучий / Організатор (Conductor)	Опис (Description)
Social analysis, Observation, Ethnographic study *(Соціальний аналіз, спостереження, етнографічне дослідження)*	Спостерігач має бути прийнятий досліджуваними людьми як «споріднена душа» і бути достатньо знайомим, щоб вони продовжували свою звичну діяльність так, ніби його там немає. (The observer must be accepted by the people being studied as a "kindred spirit" and must be sufficiently familiar that they carry on with their normal practices as if he was not there.)	Спостерігач проводить певний період часу в суспільстві чи культурі, здійснюючи детальне спостереження за всіма їхніми практиками. + Початкове розуміння системи та предметної області застосування (+ Initial understanding of the system and the application domain) + Детальне розуміння соціальних/організаційних культур, умов роботи (взаємодія в команді та спільна робота) і робочого процесу (+ Detailed understanding of social/organizational cultures, work setting (team interaction and collaborative work), and work flow) + Інформація, релевантна для проєктних рішень (+ Information relevant to design solutions)
Protocol analysis [8, 21] *(Протокольний аналіз)*	Спостерігач має бути прийнятий досліджуваними людьми як «споріднена душа» і бути достатньо знайомим... *(продовження умов з верхньої комірки)*	Суб'єкт виконує певне завдання і при цьому одночасно говорить вголос та пояснює свої думки. + Проблеми взаємодії в існуючих системах (+ Interaction problems in existing systems) + Робочий контекст і робочий процес (+ Work context and work flow)

Метод спостереження надає засоби для розвитку глибокого розуміння області застосування шляхом спостереження за діяльністю людини. Окрім неявних вимог, деякі вимоги є очевидними для зацікавлених сторін, але їх важко вербалізувати. Ми називаємо їх неявними вимогами [30]. Вербальна комунікація часто безпорадна при зборі неявних вимог. Тому спостереження за тим, як люди виконують свою рутинну роботу, є засобом отримання інформації, яку важко вербалізувати. Методи цієї групи включають етнографічні дослідження, як показано в таблиці 2.2.

Спостережні методи, як видається, добре підходять у тих випадках, коли

людям важко чітко сформулювати свої потреби, а також коли аналітики прагнуть краще зрозуміти контекст, у якому буде використовуватися бажаний продукт [31]. Прикладами неявної інформації є рутинна робота, яку люди виконують щодня інтуїтивно, та організаційні чи соціальні контексти, які потенційно впливають на вимоги. Оскільки люди знайомі з контекстом та ситуацією своєї роботи, вони свідомо не думають про рутину та робоче середовище. Їм важко пояснити, як виконується робота, хоча рутинну роботу іноді легко продемонструвати іншим [32]. Тому занурення в реальну робочу ситуацію для отримання спостережливих даних може допомогти інженерам глибоко зрозуміти схему роботи, соціальну групу, організацію та ширший контекст, у якому використовується продукт.

Оскільки методи спостереження належать до категорії поздовжніх досліджень, вони, загалом, займають більше часу, ніж інші методи, що є основним недоліком таких методів, особливо коли проект має стислий графік на етапі вимог. Крім того, спостереження вимагає чутливості та реагування на фізичне середовище. Спостерігачам легко сприйняти повну картину робочого контексту, але зазвичай важко визначити та проаналізувати їхнє сприйняття.

Крім того, методи спостереження використовуються для розуміння складних суспільств, а не для винесення суджень про те, як можна покращити або підтримати способи роботи [33]. Вони корисні для виявлення основних аспектів рутинного порядку, таких як типова схема роботи, та надають інформацію, найбільш релевантну для розробки рішень. Тому часто гарною практикою є початок з методу спостереження, щоб отримати початкове розуміння бажаного продукту, коли команда розробників не має досвіду розробки продукту в певній галузі.

Аналітичні методи

Використовуючи розмовні методи або методи спостереження, вимоги безпосередньо витягуються з поведінки людей та їхніх вербалізованих думок. Крім того, знання, що маються на увазі, хоча й не виражені безпосередньо, такі як знання експертів або інформація про нормативні акти чи застарілі продукти,

також надають інженерам багату інформацію щодо продукту. Аналітичні методи надають способи дослідження існуючої документації або знань та отримання вимог із серії висновків. Вони проілюстровані в таблиці 2.3.

Різноманітність документації може пролити світло на вимоги до бажаного продукту. Вона включає аналіз проблем, організаційні схеми, стандарти, посібники користувача існуючих систем, звіт про дослідження конкурентних систем на ринку тощо. Вивчаючи її, інженери збирають інформацію про область застосування, робочий процес, функції продукту та зіставляють її зі специфікацією вимог. Крім того, вони визначають та повторно використовують вимоги зі специфікації застарілих або подібних продуктів. Завжди варто досліджувати та шукати звіти та записану інформацію, що стосується бажаного продукту [34].

В аналітичних методах методи картування корисні для отримання знань. Як обговорювалося, багатовимірне масштабування дозволяє користувачам отримати концептуальну структуру, когнітивне картування – ідентифікувати фактори та визначати причинно-наслідкові зв'язки завдання чи процесу, а дисперсійний аналіз – використовувати існуючу систему як основу для визначення нових системних вимог. Ці методи зазвичай розглядаються як методи отримання знань, але також можна адаптувати для виявлення вимог.

Замість повторного використання вимог, досліджень документації та пов'язаних з ними методів зіставлення, де документація є основним джерелом вимог, інформація, отримана зі знань та досвіду експертів, утворює ще одне джерело вимог в аналітичних методах. Вимоги можна витягти зі знань експертів у предметній області.

Таблиця 2.3.

Перелік аналітичних методів

Метод (Method)	Джерело знань (Source)	Опис (Description)
Requirement reuse *(Повторне використання вимог)*	Документація (Documentation)	Повторне використання глосаріїв та специфікацій успадкованих (існуючих) систем або систем у межах одного сімейства продуктів для визначення вимог до бажаної системи. + Вимоги до предметної області + Характеристики інтерфейсу користувача + Організаційна політика, стандарти, законодавство тощо
Documentation studies / content analysis *(Вивчення документації / контент-аналіз)*	Документація (Documentation)	Загальноприйнятий метод, який полягає в читанні та вивченні наявної документації з метою пошуку інформації, що є релевантною та корисною для завдань виявлення вимог. + Організаційна політика, стандарти, законодавство тощо + Ринкова інформація + Специфікація успадкованих (існуючих) систем
Laddering *(Метод «драбини»)*	Знання експерта (Expert's knowledge)	Передбачає створення, перегляд та модифікацію ієрархічного змісту знань експерта, часто у формі «драбин» + Організаційна культура + Знання предметної області
Card sorting *(Сортування карток)*	Знання експерта (Expert's knowledge)	Експерта просять розсортувати за групами набір карток, на кожній з яких написано або зображено назву певної сутності предметної області. + Знання предметної області
Repertory grid *(Репертуарні решітки)*	Знання експерта (Expert's knowledge)	Зацікавлену сторону (стейкхолдера) просять указати атрибути, застосовні до набору сутностей, та значення для комірок у матриці «сутність-атрибут».

Як показано в таблиці 2.3, сходи використовуються для отримання пояснень та роз'яснень технічних термінів або суб'єктивних термінів, а також для з'ясування того, як експерти структурують свої знання про предметну область, а сортування карток та репертуарна сітка забезпечують способи виявлення атрибутів, які експерт не може одразу та легко сформулювати.

Загалом, аналітичні методи не є життєво важливими для виявлення вимог, оскільки вимоги збираються опосередковано з інших джерел, а не від кінцевих користувачів та замовників. Однак вони утворюють додаткові методи для підвищення ефективності та результативності виявлення вимог, особливо коли інформація зі застарілих або пов'язаних продуктів може бути використана повторно [35].

Синтетичні методи

Жоден окремий метод не є достатнім для розробки вимог. Враховуючи контекст та обставини, різні методи можна вибрати на різних сесіях виявлення потреб, навіть протягом однієї сесії. Наприклад, часто гарною ідеєю є почати з неформального інтерв'ю з відкритим кінцем або вивчення документації перед тим, як аналітик розпочне етнографічне дослідження. Поєднання відкритого інтерв’ю та етнографічних досліджень допомагає інженеру виявити основні аспекти та отримати загальне уявлення про сферу застосування, що сприяє проведенню подальшого етнографічного дослідження.

На відміну від поєднання окремих методів, синтетичний метод утворює цілісне ціле шляхом систематичного поєднання розмови, спостереження та аналізу в єдині методи. Аналітики та представники зацікавлених сторін спілкуються та координують свої дії різними способами, щоб досягти спільного розуміння бажаного продукту. Їх також називають методами співпраці [36]. Приклади синтетичних методів наведено в таблиці 2.4.

Синтетичні методи поєднують різні канали зв'язку та надають моделі для демонстрації функцій та взаємодії системи. Вони забезпечують гарні підказки для розпізнавання вимог у вигляді насичених семантичних моделей.

Таблиця 2.4.

Перелік методів синтезу

Метод (Method)	**Ведучий / Організатор (Conductor)**	**Опис (Description)**
Scenarios, passive storyboards *(Сценарії, пасивні розкадровки)*	Аналітики та представники зацікавлених сторін спілкуються та координують свої дії для досягнення спільного розуміння вимог. (Analysts and stakeholder representatives communicate and coordinate to reach a common understanding of the requirements.)	Це інтерактивна сесія для опису послідовності дій та подій для конкретного випадку якогось загального завдання, яке система має виконати. + Уточнені системні вимоги, пов'язані з процедурами та потоками даних завдання. + У ситуації високої невизначеності — ефективний і відносно недорогий спосіб розробити початковий набір вимог.
Prototyping, Interactive storyboards *(Прототипування, інтерактивні розкадровки)*	Аналітики та представники зацікавлених сторін спілкуються та координують свої дії... *(продовження з верхньої комірки)*	Забезпечує зацікавлених сторін конкретною (хоча й частковою) моделлю або системою, яку вони можуть очікувати отримати наприкінці проєкту. Часто використовується для виявлення та валідації системних вимог. + Функції продукту та детальні специфікації — ранній і реалістичний погляд на те, що було реалізовано.
JAD/RAD sessions *(Сесії JAD/RAD)*	Аналітики та представники зацікавлених сторін спілкуються та координують свої дії... *(продовження з верхньої комірки)*	Абревіатура означає «Спільна розробка додатків» / «Швидка розробка додатків» (Joint Application Development / Rapid Application Development) і наголошує на залученні користувачів через групові сесії з неупередженим фасилітатором.
Contextual inquiry *(Контекстне дослідження)*	Аналітики та представники зацікавлених сторін спілкуються та координують свої дії... *(продовження з верхньої комірки)*	Це поєднання відкритого інтерв'ю, спостереження на робочому місці та прототипування. Цей метод насамперед використовується для проєктування інтерактивних систем, де дизайн інтерфейсу користувача є критично важливим.

Наприклад, прототипи надають користувачам початкову версію системи, яка може нагадати їм про детальні функції, які часто ігноруються. Розкадровка – це метод між сценаріями та прототипуванням. Вона пропонує континуум можливостей, починаючи від зразків виводу до інтерактивних демонстрацій у реальному часі.

Замість методів, обмежених на етапі вимог, синтетичні методи часто використовуються на інших етапах життєвий цикл розробки продукту. Оскільки метою синтетичних методів є покращення комунікації між розробниками та замовниками, вони підходять для різних етапів процесу розробки. Вони ефективно гармонізують етап вимог з рештою заходів з розробки.

2.2 Процес розробки та документування вимог

У цьому підрозділі представлено процес розробки вимог, орієнтуючись на решту 5 тем і показуючи, як розробка вимог узгоджується із загальним процесом розробки програмного забезпечення.

Мета цієї підтеми - забезпечити розуміння того, що процес розробки вимог:

- не є окремою діяльністю фронтенду життєвого циклу програмного забезпечення, а радше процесом, який ініціюється на початку проекту та продовжує вдосконалюватися протягом життєвого циклу процесу розробки програмного забезпечення;
- повинні ідентифікувати вимоги як елементи конфігурації та керувати ними в тому ж режимі конфігурації, що й іншими продуктами процесу розробки;
- потрібно буде адаптувати до організації та контексту проекту.

Зокрема, ця підтема стосується того, як налаштовуються дії з виявлення, аналізу, специфікації, валідації та управління для різних типів проектів та обмежень. Ця підтема також охоплює дії, що забезпечують внесок у процес розробки вимог, такі як маркетинг та техніко-економічні обґрунтування.

У цій підтемі розглядаються ролі людей, які беруть участь у процесі

розробки вимог. Розробка вимог є фундаментально міждисциплінарною, і інженер вимог повинен бути посередником між сферами користувача та розробки програмного забезпечення. Часто, окрім інженера вимог, задіяно багато людей, кожен з яких має свою частку в системі. Зацікавлені сторони будуть різними для різних проектів, але завжди включають користувачів/операторів та замовника (які не обов'язково повинні бути однаковими) [37]. Це не обов'язково повинні бути однорідні групи, оскільки може бути багато користувачів і багато замовників, кожен з яких має різні проблеми. Також можуть бути інші зацікавлені сторони, які є зовнішніми по відношенню до організації користувача/замовника, такі як регуляторні органи, чиї вимоги необхідно ретельно проаналізувати. Розробники систем/програмного забезпечення також є зацікавленими сторонами, оскільки вони мають законний інтерес у отримання прибутку від системи. Знову ж таки, це може бути гетерогенна група, в якій (наприклад) системний архітектор має різні потреби, ніж системний тестувальник.

Неможливо повністю задовольнити вимоги кожної зацікавленої сторони, і завдання інженера з вимог полягає в тому, щоб домовитися про компроміс, який буде прийнятним як для основних зацікавлених сторін, так і в рамках бюджетних, технічних, регуляторних та інших обмежень. Передумовою для цього є визначення всіх зацікавлених сторін, аналіз характеру їхньої «частки» та виявлення їхніх вимог.

Підтримка та управління процесами.

У цій підтемі представлено ресурси управління проектами, необхідні та споживані процесом розробки вимог. Ця тема лише встановлює контекст для теми 3 (Ініціація та визначення обсягу) ключових знань управління програмним забезпеченням. Її основна мета полягає у встановленні зв'язку між процесами, та питаннями вартості, людських ресурсів, навчання та інструментів.

У таблиці 2.5 показано зв'язки із спільними темами в інших ключових аспектах знань.

Якість та вдосконалення процесів.

Таблиця 2.5

Зв'язок якості процесу з іншими ключовими факторами (KA)

Зв'язки із загальними темами (Links to common themes)	**Опис (Description)**
Quality *(Якість)*	Підтема якості та вдосконалення процесів пов'язана з якістю. Вона містить посилання на стандарти SPI (вдосконалення процесів програмного забезпечення), такі як моделі зрілості можливостей інженерії програмного забезпечення та систем, майбутній стандарт ISO/IEC 15504 та настанову ISO 9001-3. Процес інженерії вимог є щонайменше периферійним (другорядним) щодо них, і єдиною працею, яка безпосередньо розглядає процеси інженерії вимог, є посібник із належної практики інженерії вимог.
Standards *(Стандарти)*	Моделі/стандарти SPI, як описано в темі якості вище. Крім того, стандарт життєвого циклу інженерії програмного забезпечення ISO/IEC 12207-1995 описує діяльність з інженерії вимог у контексті первинних, підтримувальних та організаційних процесів життєвого циклу програмного забезпечення.
Measurement *(Вимірювання)*	На рівні процесів метрики вимог, як правило, є відносно крупнозернистими (узагальненими) і стосуються (наприклад) підрахунку кількості вимог, а також кількості та наслідків змін у вимогах. Якщо вони вказують на можливості для вдосконалення (що неминуче й станеться), можна виміряти ступінь і суворість використання «належної практики» вимог у процесі. Ці показники можуть слугувати для виявлення слабких місць у процесах, які мають стати ціллю для подальших зусиль із покращення.
Tools *(Інструменти)*	Загальні інструменти управління проєктами. Див. область знань (KA) з управління програмним забезпеченням.

Ця підтема стосується оцінки якості процесу інженерії вимог. Її мета - підкреслити ключову роль, яку відіграє інженерія вимог з точки зору вартості, своєчасності та задоволеності клієнтів програмними продуктами [38]. Це допоможе зорієнтувати процес інженерії вимог зі стандартами якості та моделями вдосконалення процесів для програмного забезпечення та систем. Якість та вдосконалення процесів тісно пов'язані зі знанням якості програмного забезпечення та знанням процесу розробки програмного забезпечення. Особливий інтерес представляють питання атрибутів та вимірювання якості

програмного забезпечення, а також визначення процесу розробки програмного забезпечення. Ця підтема охоплює:

- охоплення вимог інженерією стандартами та моделями вдосконалення процесів;
- заходи з розробки вимог та бенчмаркінг;
- планування та впровадження покращень;

Виявлення вимог

Ця тема охоплює те, що іноді називають «захопленням вимог», «виявленням вимог» або «отриманням вимог». Вона стосується походження вимог та того, як їх може зібрати інженер з вимог. Виявлення вимог – це перший етап у формуванні розуміння проблеми, яку має вирішити програмне забезпечення. Це, по суті, людська діяльність, під час якої визначають зацікавлені сторони та встановлюють стосунки між командою розробників (зазвичай у особі інженера з вимог) та замовником.

Джерела вимог

У типовій системі буде багато джерел вимог, і важливо, щоб усі потенційні джерела були ідентифіковані та оцінені з точки зору їхнього впливу на систему. Ця підтема розроблена для підвищення обізнаності про різні джерела вимог та структури для їх управління. Основні моменти, що розглядаються:

♦ Цілі. Термін «Мета» (іноді його називають «бізнес-занепокоєнням» або «критичним фактором успіху») стосується загальних, високорівневих цілей системи. Цілі забезпечують мотивацію для системи, але часто сформульовані нечітко. Інженери вимог повинні звертати особливу увагу на оцінку цінності (відносно пріоритету) та вартості цілей. Техніко-економічне обґрунтування є відносно недорогим способом зробити це [39].

- Знання предметної області. Інженер з вимог повинен здобути або мати доступні знання про предметну область застосування. Це дозволяє йому робити висновки про неявні знання, які зацікавлені сторони не висловлюють, оцінювати компроміси, які будуть необхідні між суперечливими вимогами, а іноді й діяти як захисник «користувача».

- Зацікавлені сторони системи. Багато систем виявилися незадовільними, оскільки вони наголошували на вимогах однієї групи зацікавлених сторін на шкоду іншим. Отже, постачаються системи, які важко використовувати або які підривають культурні чи політичні структури організації-замовника. Інженер з вимог повинен визначати, представляти та керувати «точками зору» багатьох різних типів зацікавлених сторін [40].
- Операційне середовище. Вимоги будуть визначатися середовищем, в якому працюватиме програмне забезпечення. Це можуть бути, наприклад, часові обмеження в системі реального часу або обмеження сумісності в офісному середовищі. Їх необхідно активно шукати, оскільки вони можуть суттєво вплинути на доцільність системи, її вартість та обмежити вибір дизайну.
- Організаційне середовище. Для підтримки бізнес-процесу потрібно багато систем, і це може бути зумовлено структурою, культурою та внутрішньою політикою організації. Інженер з вимог повинен бути чутливим до них, оскільки, загалом, нові програмні системи не повинні змушувати вносити незаплановані зміни до бізнес-процесу.

Методи виявлення

Коли джерела вимог визначено, інженер з вимог може почати отримувати від них вимоги. Ця підтема зосереджена на методах, які допомагають зацікавленим сторонам сформулювати свої вимоги. Це дуже складна область, і інженер з вимог повинен бути чутливим до того факту, що (наприклад) користувачам може бути важко описати свої завдання, вони можуть не вказати важливу інформацію або не бажати чи не мати змоги співпрацювати. Особливо важливо розуміти, що отримання вимог не є пасивною діяльністю, і що навіть за наявності зацікавлених сторін, які готові співпрацювати та чітко висловлювати свої вимоги, інженер з вимог повинен наполегливо працювати, щоб отримати правильну інформацію. Буде розглянуто низку методів, але основними з них є:

- Співбесіди. Співбесіди – це «традиційний» спосіб виявлення вимог. Важливо розуміти переваги та обмеження співбесід, а також те, як їх слід проводити.

- Сценарії. Сценарії цінні для забезпечення контексту для виявлення вимог користувачів. Вони дозволяють інженеру вимог створити основу для питань щодо завдань користувачів, дозволяючи ставити питання типу «що, якщо?» та «як це зробити?». Існує посилання (концептуальне моделювання), оскільки останні нотації моделювання намагалися інтегрувати нотації сценаріїв з методами об'єктно-орієнтованого аналізу.

- Прототипи. Прототипи є цінним інструментом для уточнення нечітких вимог. Вони можуть діяти подібно до сценаріїв, забезпечуючи контекст, у якому користувачі краще розуміють, яку інформацію їм потрібно надати. Існує широкий спектр методів прототипування, які варіюються від паперових макетів дизайну екранів до бета-тестових версій програмних продуктів. Існує сильний збіг з використанням прототипів для перевірки вимог.

- Зустрічі з фасилітацією. Мета цих зустрічей полягає в тому, щоб спробувати досягти підсумкового ефекту, завдяки якому група людей може краще зрозуміти свої вимоги, ніж працюючи індивідуально. Вони можуть проводити мозковий штурм та вдосконалювати ідеї, які може бути важко виявити за допомогою (наприклад, інтерв'ю). Ще однією перевагою є те, що суперечливі вимоги виявляються на ранній стадії таким чином, що зацікавлені сторони можуть розпізнати, де є конфлікт. У найкращому випадку цей метод може призвести до більш багатшого та послідовного набору вимог, ніж це було б можливо досягти в іншому випадку. Однак зустрічі потрібно проводити обережно (звідси потреба в фасилітаторі), щоб запобігти ситуації, коли критичні здібності команди підриваються груповою лояльністю, або вимоги відображають занепокоєння кількох голосних (і, можливо, старших) людей на шкоду іншим.

Спостереження. Важливість контексту систем в організаційному середовищі призвела до адаптації методів спостереження для виявлення вимог. Інженер з вимог дізнається про завдання користувачів, занурюючись у середовище та спостерігаючи за тим, як користувачі взаємодіють зі своїми системами та один з одним. Ці методи є відносно новими та дорогими, але

повчальними, оскільки вони ілюструють, що багато завдань користувачів та бізнес-процесів є занадто тонкими та складними, щоб їхні учасники могли їх легко описати.

Таблиця 2.6

Зв'язок методів виявлення з іншими ключовими факторами

Зв'язки із загальними темами	Опис (Description)
Якість	Якість виявлення вимог має прямий вплив на якість продукту. Критично важливими завданнями є розпізнавання відповідних джерел, прагнення уникнути пропуску важливих вимог та точне представлення (звітування) цих вимог.
Вимірювання	Було проведено дуже мало робіт з вимірювання процесу виявлення вимог.

2.3 Аналіз і валідація вимог до програмного забезпечення

Ця підтема стосується процесу аналізу вимог до:

- виявляти та вирішувати конфлікти між вимогами;
- виявити межі системи та те, як вона повинна взаємодіяти зі своїм оточенням;
- детальні системні вимоги до вимог до програмного забезпечення.

Традиційний погляд на аналіз вимог полягав у зведенні його до концептуального моделювання з використанням одного з методів аналізу, таких як SADT або OOA. Хоча концептуальне моделювання є важливим, ми включаємо класифікацію вимог, щоб допомогти знайти компроміси між вимогами (класифікація вимог), та процес встановлення цих компромісів (узгодження вимог) [28].

Класифікація вимог

Існує сильний збіг між класифікацією вимог та атрибутами вимог. Вимоги можна класифікувати за низкою вимірів. Приклади включають:

- Чи є вимога функціональною чи нефункціональною.
- Чи вимога походить від однієї або кількох вимог високого рівня, емерджентної властивості, чи знаходиться на високому рівні та безпосередньо накладається на систему зацікавленою стороною чи якимось іншим джерелом.
- Чи вимога стосується продукту, чи процесу. Вимоги до процесу обмежують, наприклад, вибір підрядника, методи розробки, які необхідно застосовувати, та стандарти, яких слід дотримуватися.
- Пріоритет вимоги. Загалом, чим вищий пріоритет, тим важливіша вимога для досягнення загальних цілей системи. Часто класифікується за фіксованою шкалою, такою як *обов'язковий, дуже бажаний, бажаний, необов'язковий* . Пріоритет часто має бути збалансований з вартістю розробки та впровадження.
- Обсяг вимоги. Обсяг стосується ступеня, до якого вимога впливає на систему та компоненти системи. Деякі вимоги, зокрема певні нефункціональні, мають глобальну сферу застосування, оскільки їх задоволення не можна віднести до окремого компонента. Отже, вимога з глобальною сферою застосування може сильно впливати на архітектуру системи та проектування багатьох компонентів, тоді як вимога з вузькою сферою застосування може пропонувати низку варіантів проектування з незначним впливом на виконання інших вимог.
- Волатильність/стабільність. Деякі вимоги змінюватимуться протягом життєвого циклу програмного забезпечення та навіть під час самого процесу розробки. Корисно, якщо можна зробити певну оцінку ймовірності зміни вимоги. Наприклад, у банківській програмі вимоги до функцій для розрахунку та зарахування відсотків на рахунки клієнтів, ймовірно, будуть стабільнішими, ніж вимога щодо підтримки певного типу рахунку, що не оподатковується. Перші відображають фундаментальну особливість банківської сфери (те, що рахунки можуть нараховувати відсотки), тоді як другі можуть бути застарілими через зміну державного законодавства. Позначення вимог, які можуть бути мінливими, може допомогти розробнику програмного забезпечення створити дизайн, який буде більш толерантним до змін.

Інші класифікації можуть бути доречними, залежно від звичайної практики організації-розробника та самого застосування.

Концептуальне моделювання

Розробка моделей проблеми є фундаментальною для аналізу вимог. Мета полягає в тому, щоб допомогти зрозуміти проблему, а не ініціювати проектування рішення. Отже, концептуальні моделі складаються з моделей сутностей з предметної області, налаштованих таким чином, щоб відображати їхні реальні зв'язки та залежності.

Існує кілька видів моделей, які можна розробити. До них належать потоки даних та керування, моделі станів, трасування подій, взаємодії з користувачами, моделі об'єктів та багато інших. Фактори, що впливають на вибір моделі, включають:

Характер проблеми. Деякі типи застосувань вимагають особливо ретельного аналізу певних аспектів. Наприклад, моделі потоку керування та станів ймовірно, будуть важливішими для систем реального часу, ніж для інформаційної системи.

- Експертиза інженера з вимог. Часто продуктивніше застосовувати нотацію або метод моделювання, з яким інженер з вимог має досвід. Однак може бути доцільним або необхідним застосувати нотацію, яка краще підтримується інструментами, нав'язана як вимога процесу або просто «краща»
- Вимоги замовника до процесу. Замовники можуть нав'язати інженеру з вимог певну нотацію або метод. Це може суперечити попередньому фактору.
- Доступність методів та інструментів. Нотації або методи, які погано підкріплені навчанням та інструментами, можуть не отримати широкого визнання, навіть якщо вони підходять для вирішення певних типів проблем.

Зауважте, що майже у всіх випадках корисно почати з побудови моделі системного контексту. Системний контекст забезпечує розуміння між цільовою системою та її зовнішнім середовищем. Це має вирішальне значення для розуміння контексту системи в її операційному середовищі та визначення її

інтерфейсів із середовищем.

Питання моделювання тісно пов'язане з питанням методів. Для практичних цілей метод – це нотація (або набір нотацій), що підтримується процесом, який керує застосуванням цих нотацій. Методи та нотації з'являються та зникають у міру моди. Об'єктно-орієнтовані нотації зараз у моді, але питання про те, яка нотація є «найкращою», рідко буває ясним. Існує мало емпіричних доказів на підтвердження тверджень про перевагу однієї нотації над іншою.

Формальне моделювання з використанням нотацій, заснованих на дискретній математиці та підданих логічним міркуванням, мало вплив у деяких спеціалізованих областях. Воно може бути нав'язане клієнтами або стандартами, або може пропонувати переконливі переваги для аналізу певних критичних функцій або компонентів.

Ця тема не має на меті «навчити» певному стилю моделювання чи нотації, а радше надати рекомендації щодо мети та наміру моделювання.

Архітектурне проектування та розподіл вимог

У певний момент необхідно вивести архітектуру рішення. Архітектурне проектування – це точка, в якій інженерія вимог перетинається з проектуванням програмного забезпечення або систем, і ілюструє, наскільки неможливо чітко роз'єднати обидва завдання [11]. Ця підтема тісно пов'язана з темою (архітектура програмного забезпечення). У багатьох випадках інженер вимог діє як системний архітектор, оскільки процес аналізу та розробки вимог вимагає визначення підсистем та компонентів, які будуть відповідальні за задоволення вимог. Це розподіл вимог – призначення відповідальності за задоволення вимог підсистемам.

Розподіл важливий для проведення детального аналізу вимог. Наприклад, після того, як набір вимог було розподілено між компонентом, його можна додатково проаналізувати, щоб виявити вимоги щодо того, як компонент повинен взаємодіяти з іншими компонентами для задоволення розподілених вимог. У великих проектах розподіл стимулює новий раунд аналізу для кожної підсистеми. Наприклад, вимоги до певної гальмівної ефективності автомобіля

(гальмівний шлях, безпека в поганих умовах руху, плавність натискання, необхідний тиск на педаль тощо) можуть бути розподілені між гальмівним обладнанням (механічними та гідравлічними вузлами) та антиблокувальною гальмівною системою (ABS). Тільки після того, як вимога до антиблокувальної системи була визначена та вимоги до неї розподілені, можна використовувати можливості ABS, гальмівного обладнання та емерджентні властивості (такі як вага автомобіля) для визначення детальних вимог до програмного забезпечення ABS.

Архітектурне проектування тісно пов'язане з концептуальним моделюванням. Відображення реальних об'єктів у обчислювальних компонентах не завжди очевидне, тому архітектурне проектування виділено як окрему підтему. Вимоги до нотацій та методів загалом однакові для концептуального моделювання та архітектурного проектування.

Узгодження вимог

Інша назва, яка зазвичай використовується для цієї підтеми, - «вирішення конфліктів». Вона стосується вирішення проблем із вимогами, коли виникають конфлікти; між двома зацікавленими сторонами, які вимагають взаємосумісних функцій, або між вимогами та ресурсами, або між можливостями та обмеженнями, наприклад рисунок 2.7 [17].

Таблиця 2.7

Зв'язки щодо узгодження вимог з іншими ключовими факторами (KA)

Зв'язки із загальними темами	**Опис (Description)**
Якість	Якість аналізу безпосередньо впливає на якість продукту. Принципово, що чим ретельнішим є аналіз, тим більше впевненості можна мати в якості програмного забезпечення.

Зв'язки із загальними темами	Опис (Description)
Вимірювання	Частковою метою аналізу є кількісне визначення необхідних властивостей. Це особливо важливо для таких обмежень, як вимоги до надійності або безпеки, де необхідно визначити відповідні міри для кількісної оцінки та перевірки вимог.

У більшості випадків інженеру з вимог нерозумно приймати одностороннє рішення, тому необхідно проконсультуватися із зацікавленою стороною (зацікавленими сторонами), щоб досягти консенсусу щодо відповідного компромісу. З договірних причин важливо, щоб такі рішення можна було простежити до замовника. Ми класифікували це як тему аналізу вимог, оскільки проблеми виникають в результаті аналізу. Однак, також можна навести вагомі підстави для того, щоб вважати це частиною валідації вимог.

2.4 Висновок по розділу

У другому розділі досліджено методи та процеси управління вимогами до програмного забезпечення. Розглянуто підходи до виявлення, збору та категоризації вимог, а також особливості їх документування на різних етапах життєвого циклу програмного продукту. Проаналізовано методи аналізу, перевірки та валідації вимог, що дозволяють забезпечити їх повноту, узгодженість і відповідність потребам замовника. Встановлено, що ефективне застосування зазначених методів сприяє зменшенню ризиків і підвищенню якості програмного забезпечення.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ МЕТОДІВ УПРАВЛІННЯ ВИМОГАМИ

3.1 Структура процесу управління вимогами

Дослідник збере матеріали, що стосуються ключових областей управління вимогами з точки зору процесу. За допомогою порівняльного дослідження буде проведено комплексний підхід до організації розробки програмного забезпечення. Схематичне зображення запропонованої організаційної структури показано на рис. 3.1.

Наша запропонована робота в основному зосереджена на:

Фаза I. Аналіз проблем продуктивності традиційного та гнучкого підходу в глобальній розробці програмного забезпечення.

Фаза II. Розробка оптимізованої моделі фреймворку для інтеграції традиційного підходу на глобальному рівні та гнучкого підходу на локальному рівні організаційної структури, що досягається на фазі I. Вирішення проблем, які часто полягають у вдосконаленні вимог, відсутніх вимогах та невідповідності між вимогами в проектах GSD.

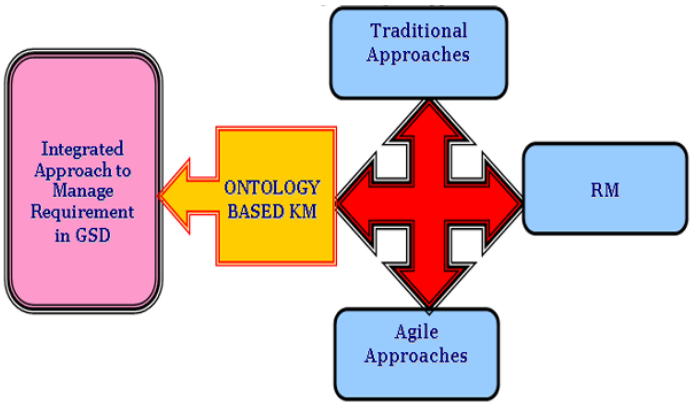

Рисунок 3.1 - Структура управління вимогами

Фаза III. Система управління знаннями на основі онтології проектування формально представляє знання предметної області та формальне представлення

вимог у GSD.

Фаза IV. Запропонувати метрики вимог для вимірювання змін вимог, середнього рівня дефектів та графіка виконання вимог у проектах GSD.

Фаза V. Сформулювати гіпотези та перевірити показники управління вимогами за допомогою статистичних методів (кореляційний та регресійний аналіз) та перевірити їх.

Управління знаннями стосується людей, щоб перевірити, як вони працюють над досягненням бізнес-цілей та цілей організації. Управління знаннями визначається як «надання потрібних знань потрібним людям у потрібний час». Застосування системи управління знаннями на основі онтології до запропонованої організаційної структури глобальної розробки програмного забезпечення охоплює багато переваг, зокрема: КМ допомагає забезпечити правильну інформацію та прийняти правильні рішення.

- Вчасно доставити систему
- Збільште прибутковість
- Покращити управління проектами, комунікацію та знання між членами команди.

КМ допомагає скоротити загальний час і вартість розробки.

Глобальний проект розробки програмного забезпечення дійсно вимагає ефективних методів управління знаннями для задоволення очікувань користувачів. Методи управління знаннями на основі онтологій відіграють значну роль в успіху географічно розподілених проектів розробки програмного забезпечення. Управління знаннями в онтологіях в основному зосереджене на обміні знаннями між глобально розподіленими командами розробників, питаннях міжкультурного обміну, питаннях комунікації (дотримання різниці в часі, часових поясах та відстанях) та технічних питаннях, таких як відсутність синхронізації.

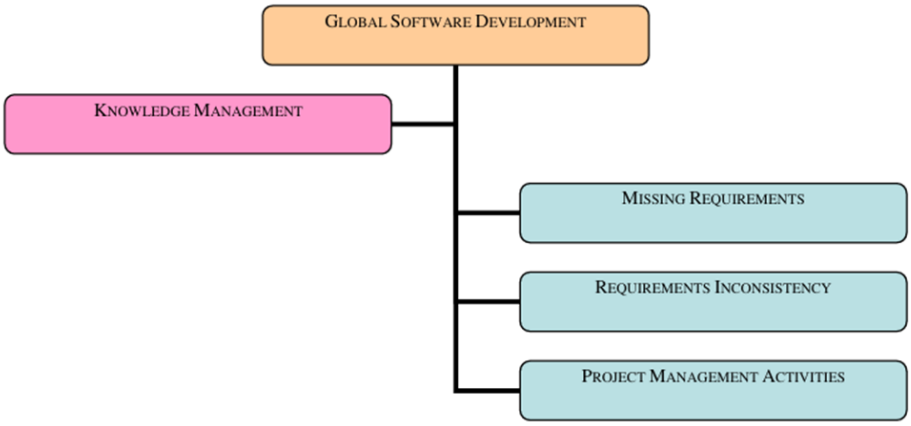

Рисунок 3.2 - Внесок методів управління знаннями у глобальну розробку програмного забезпечення

Наша головна мета - успішно керувати цими проблемами та вирішувати їх у розподіленому середовищі розробки програмного забезпечення. Запропонована нами модель створює ефективну організаційну структуру в сучасній практиці індустрії програмного забезпечення.

Глобальні проекти розробки програмного забезпечення стикаються з низкою проблем, таких як різниця в часі, мовна різноманітність, культурні проблеми, комунікація та проблеми управління знаннями. Передача та обмін знаннями нелегкі, коли члени команди працюють на різних сайтах розробки. Спільне розуміння проблемної області має вирішальне значення, коли члени команди не мають спільної мови. У багатосайтовій розробці програмного забезпечення люди обмінюються знаннями та спілкуються спільною мовою (англійською), але зацікавлені сторони походять з різних країн та регіонів, тому ідіоматичні відмінності є однією з головних проблем комунікації в GSD. Наприклад, люди зі Сполучених Штатів та Англії використовують англійську як рідну мову, але спосіб вимови термінів може відрізнятися від одного до іншого, і багато слів мають різні значення. Тому критично важливо знайти спільне розуміння проблемної області. Тому наша головна мета - подолати проблеми управління знаннями та комунікації, щоб мінімізувати ідіоматичні відмінності, і ми використовуємо систему управління знаннями на основі онтологій. Коли

члени команди походять з різних куточків країни, обмін спільною термінологією та концепціями щодо проблемної області дуже складний, тому наша мета - використовувати онтологію предметної області для вирішення цих проблем комунікації та управління знаннями. Наша запропонована система управління знаннями на основі онтологій дає чітке уявлення про проблемну область та пов'язані з нею концепції та терміни, що стосуються проблеми.

- Онтології надають спеціалізований словник представлень для процесу розробки програмного забезпечення, усуваючи концептуальні та термінологічні невідповідності.
- Використання онтологій та методів вирівнювання дозволяє вирішувати проблеми сумісності без необхідності зміни існуючих моделей.
- Онтології можуть допомогти розробити контрольні показники процесу розробки програмного забезпечення шляхом збору даних в Інтернеті та використання семантичної мережі.
- Онтологія дозволяє як передавати знання, так і спрощувати цикл розробки від проекту до проекту.
- Онтології сприяють спільному розумінню серед розробників програмного забезпечення, а також використовуються як моделі предметної області.
- Онтології спрощують процес отримання знань, використовуючи однакову концептуалізацію для різних програмних застосунків.
- Онтології дозволяють зменшити термінологічні та концептуальні невідповідності, змушуючи різних користувачів обмінюватися інформацією та обмінюватися інформацією під час онтологічного аналізу.
- Онтології також забезпечують удосконалену комунікацію між інструментами, що входять до складу середовища.

Онтології, коли вони є машинно-зрозумілими представленнями, допомагають у розробці інструментів для діяльності з розробки програмного забезпечення.

Згідно з опитуванням, у Великій Британії було проведено 1027 проектів, з яких 13% не зазнали невдачі, а каскадна модель була одним з «головних факторів, що сприяли невдачі», причому проблема була вирішена у 82% проектів. Проекти Міністерства оборони США дійшли висновку, що «46% систем дійсно не відповідали реальним потребам, а також вимогам, визначеним користувачем, вони ніколи не були успішно використані, а ще 20% проектів потребують значної переробки», щоб бути придатними для використання. Аналізний центр програмного забезпечення та дослідники розробки програмного забезпечення в Ізраїлі заявили, що гнучкі методи на 29% кращі за вартістю, на 91% кращий графік, на 97% краща продуктивність, на 50% краща якість, на 400% краща задоволеність та на 470% краща рентабельність інвестицій порівняно з традиційними підходами. В опитуванні зазначено, що традиційні підходи покращують якість та продуктивність програмної системи, якщо гнучкий підхід задовольняє реальні потреби клієнтів та успіх проекту. Вимоги постійно змінюються під час виконання проекту. Вимоги змінюються через підвищення ефективності системи для задоволення реальних потреб користувачів. Традиційними підходами важко реагувати на зміни замовника після прийняття вимог, але гнучкий підхід полягає в динамічному впровадженні змінних вимог, навіть якщо проект просувається. Запропонована нами модель полягає в створенні змішаної організаційної структури, що поєднує традиційні підходи на глобальному рівні для остаточного визначення вимог до розробки системи та гнучкий підхід на локальному рівні для динамічного виявлення, аналізу та перегляду вимог до програмного забезпечення. Причиною використання цієї змішаної організаційної структури є те, що в наш час традиційний спосіб розробки програмного забезпечення зіткнувся з кількома викликами/проблемами, такими як розуміння змінних вимог у багатосайтовому середовищі, гнучкий підхід успішно керує викликами в глобальному середовищі розробки програмного забезпечення. Щоб впровадити як традиційний, так і гнучкий методи в організації, використати переваги цих двох методів для підвищення точності, продуктивності та якості глобальних проектів розробки

програмного забезпечення. Застосувати практики управління знаннями на основі онтологій для обох підходів для вирішення проблем управління вимогами, таких як відсутні вимоги та невідповідності, проблеми комунікації та управління знаннями, а також покращити діяльність з управління проектами в глобальних проектах розробки програмного забезпечення. Ми пропонуємо метрики управління вимогами для вимірювання змін вимог, середньої частоти дефектів та графіка вимог, щоб підвищити ефективність процесу розробки програмного забезпечення.

В опитуванні зазначено, що гнучка розробка програмного забезпечення приносить багато переваг розробникам-розробникам, зокрема:

- Раннє та безперервне постачання програмної системи.
- Динамічно враховуйте зміни у вимогах, навіть прогрес у розробці.
- Більше про залучення клієнтів та кінцевих користувачів до розробки системи.
- Окупність інвестицій якомога швидше.
- Відповідати очікуванням зацікавлених сторін.
- Доступна простота та видимість фактичного прогресу проекту.
- Розмова віч-на-віч.
- Зниження ризиків під час загальної розробки програмної системи.

Вимога – це потреба замовника. Вимоги – це можливості та намір, яким повинен відповідати будь-який продукт чи послуга, що є спільним для всієї розробки системи та інших інженерних видів діяльності. Термін «інженерія» також означає, що результати процесу інженерії вимог повинні бути ретельно спроектовані, де ці «результати» зазвичай розуміються як детальні специфікації.

Інженерія вимог визначається наступним чином: «Інженерія вимог (ІВ) – це сукупність дій, пов’язаних з визначенням та повідомленням мети програмно-інтенсивної системи та контексту, в якому вона буде використовуватися. Отже, ІВ діє як місток між реальними потребами користувача, клієнта та інших зацікавлених сторін, на яких впливає програмна

система, та можливостями та перспективами, на які впливають програмно-інтенсивні технології».

Таблиця 3.1.

Показники, визначені та перевірені в цій роботі

Тип показників (Type of Measures)	**Атрибути (Attributes)**	**Сутність (Entity)**	**Теоретична / Емпірична валідність (Theoretical /Empirical Valid)**	**Середовище (Environment)**
Загальна кількість вимог, кількість початкових, поточних та кінцевих вимог, статус вимог, кількість змін на одну вимогу, статус, тип, причина та вартість зміни вимог	Розмір та статус специфікацій вимог, розмір та статус змін до вимог	Управління вимогами	Репрезентативна теорія Кітченхема	Академічне кейс-стаді
Кількість рядків, слів, акторів, прецедентів (use cases), суб'єктивна мінливість, мінливість (кількість змін: значні, помірні, незначні, редакційні зміни)	Специфікації вимог	-	Кореляція Спірмена	Промислове програмне забезпечення
Показники стабільності (обсяг інформації, що міститься у вимогах у момент часу t) та ефективності	Обсяг доопрацювання (переробки), суб'єктивно сприймана ефективність роботи	Процес аналізу вимог	Графік (діаграма) даних	Академічний експеримент
Кількість змін у вимогах, середня частота виникнення дефектів, графік (план) виконання вимог	Зміни у вимогах, зміщення графіка та частота виникнення дефектів	Управління вимогами	Коефіцієнт кореляції Пірсона та регресійний аналіз	Вебпроєкт

Управління вимогами визначається наступним чином: «Управління вимогами – це набір дій, які допомагають команді проекту визначати, контролювати та відстежувати вимоги та зміни до вимог у будь-який час у процесі виконання проекту *».*

Таблиця 3.2

Заходи з управління вимогами

№ п/п (S.No)	**Тип вимірювання (Measurement Type)**	**Показник / Міра (Measure)**	**Сутність (Entity)**
1.	Прямий (Direct)	Вимоги, на які вплинули зміни (Requirements affected by Change)	Процес управління вимогами (Requirement management process)
2.	Прямий (Direct)	Суперечливі вимоги (Inconsistent requirements)	Процес управління вимогами (Requirement management process)
3.	Прямий (Direct)	Пропущені вимоги (Missing requirements)	Процес управління вимогами (Requirement management process)
4.	Прямий (Direct)	Неповні вимоги (Incomplete requirements)	Процес управління вимогами (Requirement management process)
5.	Прямий (Direct)	Початкові вимоги (Initial requirements)	Процес управління вимогами (Requirement management process)
6.	Прямий (Direct)	Кінцеві вимоги (Final requirements)	Процес управління вимогами (Requirement management process)
7.	Непрямий (Indirect)	Графік (план) виконання вимог (Requirement schedule)	Процес управління вимогами (Requirement management process)

Метрики програмного забезпечення, що використовуються в процесі управління вимогами, використовуються для надання інформації, необхідної для прийняття ключових рішень щодо проекту та вжиття відповідних заходів.

Метрики використовуються для постійного вдосконалення та контролю проекту в процесі управління вимогами. Загальний та комплексний набір заходів управління вимогами визначено для реалізації цілей KPA (ключової області процесу) управління вимогами в рамках SW-CMM рівня2, що зазначено в таблиці 3.2.

3.2 Оцінка інструментів управління вимогами

Більшість досягнень у сфері інструментів та методів розробки програмного забезпечення за останні три десятиліття були зосереджені на продуктивності, а не на якості, і в першу чергу стосувалися роботи окремих осіб [13]. Зараз потрібен засіб підвищення продуктивності та якості роботи, що виконується командами. Часто ці команди розосереджені на географічно розподілених місцях.

Продажі інструментів управління вимогами неухильно зростають останніми роками [15]. На ринку зараз представлено так багато інструментів, які обіцяють підтримку процесу управління вимогами (або його частини), що проведення детальної оцінки всіх з них є не доцільним і неможливим. Тому для проведення змістовної оцінки необхідно спочатку визначити основні вимоги до інструменту для спільного управління вимогами, а потім оцінити характеристики комерційних інструментів управління вимогами відповідно до цих вимог.

Вимоги до інструменту для спільного управління вимогами

Основні вимоги до інструменту управління вимогами добре визначені в літературі [9,16]. З огляду на акцент цього дослідження на підтримці міждисциплінарної, розподіленої співпраці, висуваються деякі додаткові вимоги. Підсумовуючи, інструмент для підтримки спільного управління вимогами до програмного забезпечення повинен мати змогу:

- підтримувати унікальні ідентифіковані описи всіх вимог;
- класифікувати вимоги за логічними групами, визначеними

користувачем;

- визначати вимоги, використовуючи текстові, графічні та модельні описи, з підтримкою мультимедійних описів (таких як зображення та анімовані симуляції);
- визначити простежувані зв'язки між вимогами;
- перевірити відповідність вимог користувача технічним вимогам проекту;
- вести журнал аудиту змін; архівувати базові версії; та задіяти механізм для автентифікації та затвердження запитів на зміни;
- підтримувати безпечну, одночасну спільну роботу між членами багатопрофільної команди, яка може бути географічно розподілена;
- підтримувати стандартні методи моделювання систем та нотації;
- вести вичерпний словник даних усіх компонентів та вимог проєкту у спільному репозиторії;
- генерувати попередньо визначені та спеціальні звіти;
- створювати документи, що відповідають стандартним промисловим шаблонам, з підтримкою якості виводу презентації, попереднього перегляду WYSIWYG та вбудованих засобів контролю якості документів;
- безперешкодно з’єднуватися з іншими інструментами та системами, підтримуючи сумісні протоколи та стандарти.

Слабкі сторони сучасних інструментів управління вимогами

З огляду на ці вимоги, автори провели огляд пропозицій постачальників. Як було описано раніше, було складено короткий список доступних на даний момент «найкращих у своєму класі» інструментів управління вимогами, який наведено в Таблиці 3.3.

Потім ці інструменти були детально оцінені, щоб визначити загальні сильні та слабкі сторони. Хоча більшість інструментів, що увійшли до короткого списку, відповідають усім або майже всім вищезазначеним вимогам з різним ступенем охоплення, було виявлено низку поширених слабких сторін, які описані нижче.

Проблеми зручності використання та розуміння для користувачів без технічних знань

Деякі інструменти, що рекламуються як інструменти управління вимогами, слід належним чином класифікувати як інструменти системної інженерії/CASE. Такі інструменти спеціально розроблені для використання кваліфікованими фахівцями, які володіють як методами програмної інженерії, так і функціональністю самого інструменту. Через складність CASE-інструментів та артефактів, які вони створюють, вони непрактичні для спілкування з нетехнічними зацікавленими сторонами [14]. Дійсно, було зазначено, що CASE-інструменти іноді можуть навіть знижувати ефективність спілкування між розробниками та кінцевими користувачами [17]. Очевидно, що орієнтація CASE-інструментів означає, що вони самі по собі неадекватні для підтримки спільного процесу управління вимогами, що включає багатопрофільні команди.

Таблиця 3.3

Провідні інструменти управління вимогами

Інструмент (Tool)	**Версія (Version)**	**Розробник / Постачальник (Vendor)**
CORE Enterprise	1.3	Vitech Corporation
DOORS	4.0	Quality Systems & Software Ltd
Caliber-RM	1.0	Technology Builders, Inc.
RDD-100	4.1.1	Ascent Logic Corporation
RequisitePro	4.0	Rational Software Corporation
icCONCEPT-RTM	3.7	Integrated Chipware
SLATE	4.1	TD Technologies, Inc.
Vital-Link	2.3	Compliance Automation, Inc.
XTie-RT	3.1	Teledyne Brown Engineering

Незбалансований вибір методів специфікації

Рекомендовано, щоб вимоги описувалися як методами, що використовуються користувачем (наприклад, природною мовою), так і методами, що використовуються розробником (наприклад, діаграмами моделей), для покращення комунікації та досягнення балансу між технічною ясністю та загальною зрозумілістю [9]. Небагато інструментів досягають цього балансу. Деякі з них базуються на документах з підтримкою наративів, анотацій та напівструктурованих описів природною мовою, але надають мало або взагалі не надають можливостей моделювання. Інші, більше схожі на інструменти CASE, мають сильні можливості моделювання, але є обмежувально суворими, оскільки вони не дозволяють використовувати природну мову або інші високоекспресивні медіа, такі як насичені зображення. Дуже мало хто підтримує складні мультимедійні документи.

Відсутність підтримки спільної роботи

Чимало інструментів орієнтовані на автономну роботу окремих користувачів, хоча важливість технологічної підтримки для скоординованої командної роботи широко визнана [14]. З тих інструментів , що підтримують спільну роботу, жоден не ідеально підходить для використання міждисциплінарною, розподіленою командою, де зацікавлені сторони мають різні навички та потреби.

Попередні дослідження показали, що автоматизовані інструменти, що підтримують скоординовану, спільну розробку програмного забезпечення, мало впроваджуються в промисловості [18]. CASE-репозиторії – це перший крок. Однак цілеспрямована співпраця вимагає вищого рівня групової комунікації, ніж просто доступ до спільного репозиторію. Швидше, має бути постійна міжособистісна взаємодія, щоб виникало спільне розуміння складних питань. Наявні наразі інструменти не пропонують достатньо потужного механізму співпраці для ефективного досягнення цієї мети.

Дизайн прототипу

Ми розглядаємо вищезгадані недоліки як найсуттєвіші перешкоди для

ефективного спільного управління вимогами. У спробі усунути ці недоліки було розроблено та створено прототип *RM-Tool*. Він базується на багатокористувацькій веб-мультимедійній базі даних, реалізованій за допомогою групового програмного забезпечення Lotus Notes. Оскільки *RM-Tool* є дослідницьким прототипом, його сфера застосування суворо обмежена. Замість амбітної спроби забезпечити повне покриття всіх раніше викладених вимог, він зосереджується переважно на тих, які наразі погано задовольняються комерційними інструментами.

Огляд можливостей RM-Tool

RM-Tool прагне керувати вимогами та контролювати їх у рамках процесу, який активно залучає міждисциплінарні, розподілені проектні команди. Ключовими цілями *RM-Tool* є:

- сприяти покращенню комунікації та розуміння вимог серед усіх зацікавлених сторін проекту шляхом забезпечення балансу між технічними та нетехнічними методами специфікації;
- сприяти посиленій співпраці між розробниками та кінцевими користувачами під час визначення та перевірки вимог;
- задіяти механізм для контролю впливу змін на вимоги.

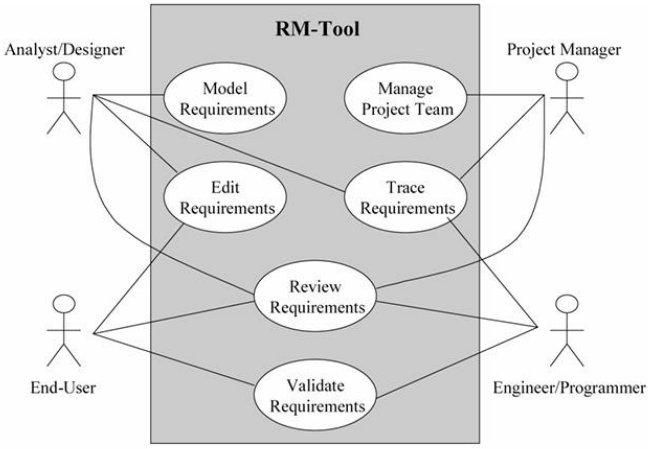

Рисунок 3.3 - Діаграма сценаріїв використання RM-Tool на загальному рівні.

Перша та друга цілі безпосередньо корелюють зі спостережуваними

недоліками сучасних інструментів управління вимогами. Третя ціль є наслідком перших двох; враховуючи додаткову динамічність спільного підходу, управління змінами у вимогах стає ще більш важливим.

На рисунку 3.3 наведено загальний огляд функцій *RM-Tool* та різних ролей, які відіграють зацікавлені сторони. Хоча зацікавлені сторони мають різні типи та рівні залученості, зауважте, що немає функції, яка б цікавила лише одну зацікавлену сторону. *RM-Tool* було розроблено з чітким урахуванням цієї розбіжності точок зору.

Ключові особливості *RM-Tool* полягають у тому, що він:

- підтримує спільний словник даних у централізованій, багатокористувацькій, мультимедійній базі даних з повним веб-доступом;

підтримує описи вимог, використовуючи як технічні, так і нетехнічні методи, з гіперпосиланнями між відповідними описами.

- забезпечує базову підтримку стандартних нотацій моделювання;
- підтримує зберігання та маніпулювання форматованими текстами та складеними мультимедійними документами, що особливо корисно для опису складних вимог та демонстрації змодельованих реалізацій;
- підтримує незалежний від часу та місця електронний зв'язок між командою зацікавлених сторін, використовуючи безпечні, автентифіковані з'єднання;
- підтримує відстеження вимог у зворотному та прямому напрямку, від джерела до реалізації;
- спрощує управління проектами, розподіляючи пріоритети, обов'язки та терміни;
- має розширені можливості звітності та може створювати паперові копії SRS.

RM-Tool не вимагає та не нав'язує жодної формалізованої методології чи парадигми. Натомість він має компоненти, здатні підтримувати процесоорієнтовані, орієнтовані на дані та/або об'єктно-орієнтовані методи (див. рис. 3.4).

Вимоги користувача є основними документами в базі даних *RM-Tool* і зазвичай описуються за допомогою орієнтованих на користувача методів, таких як природна мова, сценарії використання, макети та діалогові блок- схеми. *RM-Tool* має редактор форматованого тексту та додатково може імпортувати дані в багатьох стандартних форматах текстових процесорів, електронних таблиць та баз даних, а також графіку, зображення, анімацію та інші типи медіа, якщо необхідно. Вимоги можна розділити на логічні, визначені користувачем групи для кращої організації та управління. За необхідності можна вивести вимоги нижчого рівня з вимог вищого рівня, а також вказати кореляції та залежності між вимогами. Для кожної вимоги вказується її джерело, метод збору, власники та уповноважені редактори.

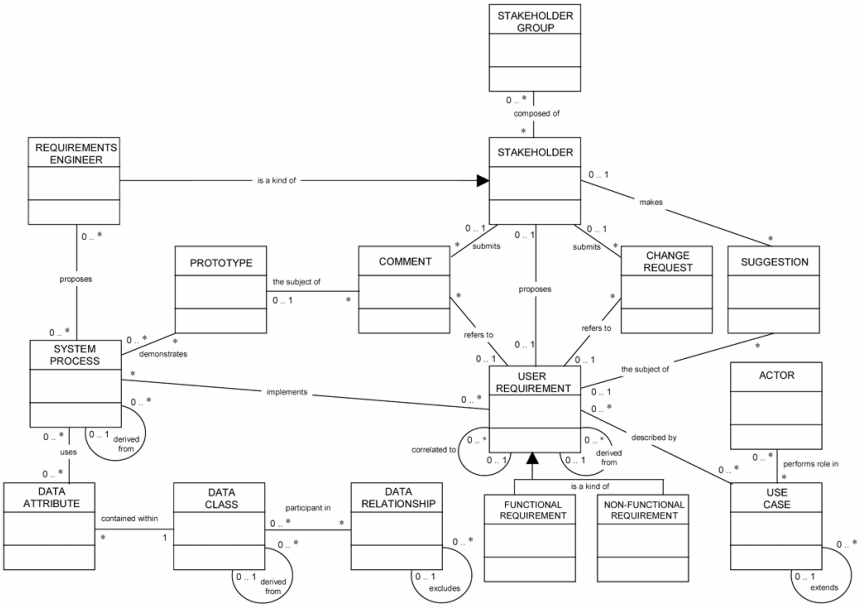

Рисунок 3.4 - Діаграма класів для RM-Tool.

Після введення вимоги користувача команда розробників може проаналізувати її, щоб визначити зусилля, використання ресурсів, технічну доцільність та цільову дату реалізації. *RM-Tool* наполягає на тому, щоб кінцеві користувачі вказували пріоритет («Обов'язковий» або «Необов'язковий») для кожної вимоги та обґрунтовували її переваги. Таким чином, зусилля розробників можуть бути пріоритетизовані та спрямовані туди, де це має найбільшу користь.

Крім того, *RM-Tool* прагне виявити та ізолювати нестабільні вимоги, фіксуючи для кожної вимоги ймовірність того, що вона може змінитися, та ймовірність того, що це може призвести до перевищення часу та витрат. Ці нестабільні вимоги та, як наслідок, інші вимоги, що походять від них або з якими вони пов’язані, можна ретельно контролювати, а їхнє впровадження можна відкласти до того часу, поки вони не стануть чіткішими та стабільнішими.

Перед етапом аналізу вимог кінцеві користувачі провели власне попереднє дослідження та підготували дві редакції черновика документа під назвою «Функціональна специфікація», підготовленого за допомогою Microsoft Word та Microsoft PowerPoint. Хоча кінцеві користувачі доклали чимало зусиль для нав'язування шаблону, документ був неповним та містив неоднозначності. Деякі розділи, включаючи « *Вступ»* та *«Обсяг проекту та припущення»,* були повністю порожніми. Здавалося, що хоча кінцеві користувачі знали, що SRS повинна містити пояснювальні попередні матеріали, вони не розуміли, що саме має бути включено та як представляти такі матеріали.

Початковий проект структури вимог, підготовлений кінцевими користувачами, був, за їхнім власним визнанням, «дуже схематичним». Вони охоче визнали, що їхня недосвідченість у написанні таких документів та використанні методів специфікації була стримуючим фактором. Проект документа містив базовий та дуже неповний опис логічних структур записів необхідних системних даних, а також макети екранів та звітів. Не було глосарію чи визначень термінів, внаслідок чого низка синонімів використовувалася заплутано та непослідовно.

Відправною точкою для застосування *RM-Tool* був аналіз чернетки документа та імпорт вимог, описаних у ньому. Потім було проведено серію особистих групових та індивідуальних співбесід між розробниками та користувачами під час раннього аналізу вимог. Результати цих співбесід були записані та введені в *RM-Tool* у вигляді нових та змінених *Вимог користувачів.* У міру внесення вимог до бази даних *RM-Tool* розробники оцінювали рівень зусиль та використання ресурсів, які знадобляться для їх реалізації. Для кожної

вимоги були визначені рейтинги ризику та стабільності, щоб вказати на ймовірність перевитрати часу та коштів або різких змін. Кінцевих користувачів попросили призначити статус для кожної вимоги: «Запропоновано», «Схвалено» або «Відхилено». Після визначення зусиль, ризику, стабільності та статусу для кожної вимоги було складено графік робіт із зазначенням цільових дат реалізації. До плану реалізації були включені лише схвалені вимоги. Вимоги, які були визначені як ризиковані або нестабільні, були відкладені на якомога пізніший термін, щоб мінімізувати будь-який побічний ефект.

RM -*Tool* використовувалися для створення структур даних та специфікацій процесів. *Вимоги користувача* були зіставлені з *системними процесами* , які, у свою чергу, були визначені за допомогою структурованої англійської мови, діаграм розташування екранів та звітів, а також блок-схем. Ці специфікації потім лягли в основу кодування та створення прототипів. Були додані гіперпосилання, що пов'язували між собою пов'язані *Вимоги користувача*, *Системні процеси* та Веб- *прототипи,* щоб можна було простежити вимогу від джерела до реалізації. Веб-прототипи складалися з анімаційних послідовностей, що демонстрували симульовані взаємодії, знімків екрана, що зображували запропоновані системні інтерфейси, прототипів перевірки введення даних за допомогою HTML-форм та Javascript, а також простих HTML-макетів системних звітів. Усі коментарі, пропозиції та запити на зміни, зроблені кінцевими користувачами щодо демонстрацій запропонованих реалізацій, реєструвалися в *RM-Tool.*

Протягом етапів проектування та кодування статус реалізації вимог («Робота в процесі», «Завершено», «Відмовлено» або «Затримано») відповідно оновлювався в *RM-Tool* . Переглядаючи звіти RM-Tool з управління проектами, розробники та користувачі могли відстежувати статус виконання відповідно до узгоджених цільових дат реалізації. Якщо виникали затримки, причини документувалися шляхом додавання *коментарів* до відповідних *Вимог користувачів.* Таким чином, будь-які затримки в часі, що виникли, були чітко видимими, як і індивідуальна відповідальність.

Управління змінними вимогами

До кінця попереднього аналізу вимог, до початку проектування, у базі даних *RM-Tool було зареєстровано 166 унікальних вимог користувачів* . До кінця проектування було зареєстровано 208 вимог. Протягом проекту було зареєстровано загалом 74 запити на зміни (див. рис. 3.5), з яких усі, крім шести, призвели до модифікацій вимог. Більшість із них були отримані під час проектування. Не дивно, що коли кінцевим користувачам показали ранні прототипи, вони почали розширювати набір вимог та змінювати існуючі вимоги. Кількість запитів на зміни зменшувалася в міру просування проекту, але знову дещо зросла під час тестування.

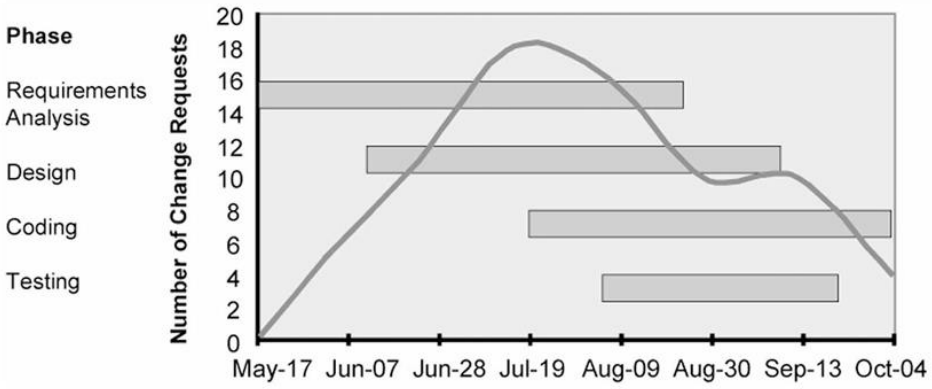

Рисунок 3.5 - Згладжений графік, що відображає частоту надходження запитів на внесення змін за етапами проекту.

Більшість запитів на зміни (31%) призвели до появи нових вимог, які не були включені або навіть не згадані в початковому проекті специфікації. Ще 28% запитів на зміни стосувалися внесення значних змін до раніше встановлених вимог, що виникли через невизначеність кінцевого користувача. Чимало з цих змін мали широкий обсяг, оскільки вони впливали на спільні функції, що були відтворені в кожному з шести окремих модулів системи. Більшість із них стосувалися проблем інтерфейсу користувача, які виникли на ранніх етапах проектування. Розробники були раді вирішити такі проблеми на цьому етапі, оскільки інакше неминуче виникло б значне невдоволення користувачів на

пізнішому етапі та, як наслідок, потреба в виснажливій переробці.

П'ятнадцять відсотків запитів на зміни стосувалися розширення вимог, визначених у початковій специфікації. Здебільшого це були незначні проблеми, такі як додавання додаткових полів на екран або у звіт. Помилки кодування під час створення прототипу призвели до появи ще 12% від загальної кількості запитів на зміни. Ще 9% стосувалися вимог, які більше не вважалися актуальними – здебільшого це були прості проблеми, такі як видалення поля з екрана або звіту, оскільки воно вважалося надмірним або зайвим.

3.3 Результати впровадження та аналіз ефективності методів управління вимогами

Як зазначалося раніше, ми вважаємо найважливішими факторами, що перешкоджають сучасним інструментам розробки вимог у їхній підтримці справді спільного процесу управління вимогами, (1) проблеми зручності використання та розуміння для нетехнічних користувачів; (2) незбалансований вибір методів специфікації; та (3) відсутність підтримки спільної роботи. Ці перешкоди мотивували розробку *RM-Tool,* цілями якого є:

- сприяти покращенню комунікації та розуміння вимог серед усіх зацікавлених сторін проекту шляхом забезпечення балансу між технічними та нетехнічними методами специфікації;
- сприяти посиленій співпраці між розробниками та кінцевими користувачами під час визначення та перевірки вимог;
- задіяти механізм для контролю впливу змін на вимоги.

У наступній частині цієї роботи ми оцінимо, якою мірою *RM-Tool* виконав ці цілі.

Очікувані результати спільного підходу

Усі зацікавлені сторони висловили думку, що високий ступінь співпраці, досягнутий завдяки застосуванню *RM-Tool* , був вигідним і призвів до таких переваг:

- Покращений обмін знаннями від розробників до кінцевих користувачів – кінцеві користувачі мали обмежені технічні знання на початку проекту. Це обмежувало їхні очікування щодо можливостей. Коли їхнє уявлення про межі технологій ставало більш обґрунтованим, очікування щодо технічної доцільності ставали більш реалістичними. Деякі очікування спочатку були наївно оптимістичними, тоді як інші були приглушеними. Тому деякі вимоги були відкинуті, оскільки кінцеві користувачі змирилися з тим, що вигода від їх впровадження буде непропорційною необхідному часу, витратам та зусиллям. Крім того, було виявлено багато «нових» вимог, коли кінцеві користувачі побачили, що можливо.
- Покращений обмін знаннями між кінцевими користувачами та розробниками – розробники спочатку погано розуміли предметну область застосування. У міру просування проекту вони краще ознайомилися з предметною областю застосування та відчували меншу дезорієнтацію.
- Реалістичні часові рамки – оскільки кінцеві користувачі та розробники разом працювали над визначенням вигоди, ризику, стабільності та зусиль для кожної вимоги, стало можливим визначити пріоритетність вимог за прогнозованою датою випуску, що було практично можливо.
- Покращена якість наданого програмного забезпечення – кінцеві користувачі вважали, що серйозність помилок протягом усього проекту була незначною. Через високий рівень комунікації між розробниками та кінцевими користувачами багато помилок було виявлено невдовзі після їх впровадження. Зокрема, механізм управління запитами на зміни був життєво важливим для успіху проекту, оскільки він виявляв та виправляв проблеми з інтерфейсом користувача, які, якщо їх залишити непоміченими, зрештою негативно вплинули б на зручність використання системи.
- Неправильним діям було запобігнуто - кінцеві користувачі також вважали, що регулярне спілкування з розробниками допомагає швидко роз'яснювати помилкові уявлення розробників, які в іншому випадку могли б призвести до марних зусиль.

Короткий огляд вражень зацікавлених сторін щодо RM-Tool

Висновки та подальша робота

Ця робота ґрунтується на аргументі, що спільний підхід до управління вимогами до програмного забезпечення є необхідним у розробці сучасних систем, щоб покращити комунікацію між зацікавленими сторонами проекту та тим самим призвести до підвищення продуктивності та якості. Через складність проектів розробки програмного забезпечення, а також тому, що зацікавлені сторони, ймовірно, географічно розподілені, використання автоматизованого інструменту для підтримки такої співпраці є важливим. З цією метою автори розробили та перевірили прототип у тематичному дослідженні.

Результати дослідження підтверджують думку, що співпраця між розробниками та користувачами може покращити обмін знаннями, виявити приховані вимоги, краще сформувати очікування, допомогти у ранньому виявленні помилок, а також покращити планування та прийняття рішень. Крім того, було зазначено, що застосування інструменту для співпраці для підтримки процесу управління вимогами може призвести до покращення управління проектами, кращого управління організаційною пам'яттю, а також кращої комунікації та розуміння. Однак переваги інструменту для співпраці, безумовно, будуть обмеженими, якщо не буде чіткого етосу якості та дисциплінованих методів роботи. Крім того, інструмент для співпраці навряд чи зможе самостійно керувати складними соціальними процесами, що характеризують розробку програмного забезпечення, тому завжди може виникнути потреба в особистому втручанні людини.

Очевидним недоліком цього дослідження є те, що воно базується на окремому випадку, і хоча часові рамки були реалістичними, команда проекту була досить невеликою. Це може бути основою для майбутньої роботи, щоб дослідити, наскільки інструменти спільного управління вимогами, такі як *RM-Tool,* є масштабованими. Ще одне питання, варте дослідження, полягає в тому, чи підходить такий підхід до спільної розробки, як описаний, для всіх категорій систем, чи, можливо, він більше підходить для бізнес-інформаційних систем.

3.4 Висновок по розділу

У третьому розділі розглянуто практичні аспекти реалізації процесів управління вимогами та оцінки їх ефективності. Проаналізовано структуру процесу управління вимогами, досліджено функціональні можливості сучасних програмних засобів підтримки цього процесу та виконано оцінку їх застосування. Результати аналізу показали, що впровадження систематизованих методів і спеціалізованих інструментів управління вимогами дозволяє покращити контроль змін, забезпечити простежуваність вимог і підвищити ефективність розробки програмного забезпечення. Отримані результати підтверджують важливість якісного управління вимогами для успішної реалізації програмних проєктів.

ВИСНОВКИ

У ході виконання бакалаврської роботи на тему «Методи управління вимогами до програмного забезпечення» було проведено комплексне дослідження теоретичних та практичних аспектів процесу управління вимогами в сучасній розробці програмного забезпечення. У роботі проаналізовано основні підходи до визначення, документування, аналізу та супроводу вимог, а також досліджено сучасні інструменти, які застосовуються для підтримки процесу управління вимогами протягом усього життєвого циклу програмного продукту. Отримані результати підтвердили важливість якісного управління вимогами як одного з ключових чинників успішної реалізації програмних проєктів.

На початковому етапі дослідження було розглянуто теоретичні основи управління вимогами до програмного забезпечення. Особливу увагу приділено класифікації вимог, принципам їх формування та ролі вимог у процесі створення програмних систем. Проведений аналіз показав, що правильне визначення та документування вимог сприяє зниженню кількості помилок під час розробки, покращенню взаємодії між учасниками проєкту та підвищенню якості кінцевого програмного продукту.

У другому розділі було досліджено методи та інструменти управління вимогами. Розглянуто процеси виявлення, аналізу, узгодження та контролю змін вимог, а також особливості їх застосування в різних моделях розробки програмного забезпечення. Проведене дослідження дозволило визначити переваги сучасних підходів до управління вимогами, серед яких простежуваність, контроль версій, можливість автоматизації процесів та підвищення ефективності командної роботи.

У третьому розділі виконано аналіз структури процесу управління вимогами та здійснено оцінку сучасних інструментів, що використовуються для підтримки цього процесу. Проведено порівняння функціональних можливостей різних програмних засобів, визначено їх сильні та слабкі сторони, а також

розглянуто практичні аспекти їх використання в реальних проєктах. Отримані результати свідчать про те, що впровадження спеціалізованих інструментів дозволяє значно підвищити якість управління вимогами та зменшити ризики виникнення помилок на етапах розробки.

У результаті виконаного дослідження встановлено, що ефективне управління вимогами є необхідною умовою створення якісного програмного забезпечення. Використання сучасних методів та інструментів управління вимогами забезпечує своєчасне виявлення та усунення суперечностей, покращує комунікацію між учасниками проєкту та сприяє успішному досягненню поставлених цілей. Отримані результати можуть бути використані під час розробки програмних систем різного призначення для підвищення ефективності процесів аналізу, документування та супроводу вимог.

У перспективі подальші дослідження можуть бути спрямовані на вивчення можливостей застосування штучного інтелекту та автоматизованих засобів аналізу для підтримки процесів управління вимогами, а також на інтеграцію сучасних підходів до управління вимогами з гнучкими методологіями розробки програмного забезпечення.

СПИСОК ПОСИЛАНЬ НА ДЖЕРЕЛА

1. Sommerville, I. (2021). *Software Engineering* (11th ed.). Pearson. https://www.pearson.com/en-us/subject-catalog/p/software-engineering/P200000003192

2. Pressman, R. S., & Maxim, B. R. (2020). *Software Engineering: A Practitioner's Approach* (9th ed.). McGraw-Hill. https://www.mheducation.com

3. Wiegers, K. E., & Beatty, J. (2013). *Software Requirements* (3rd ed.). Microsoft Press. https://www.microsoftpressstore.com

4. Robertson, S., & Robertson, J. (2012). *Mastering the Requirements Process* (3rd ed.). Addison-Wesley. https://www.informit.com

5. Pohl, K. (2010). *Requirements Engineering: Fundamentals, Principles, and Techniques*. Springer. https://link.springer.com/book/10.1007/978-3-642-12578-9

6. ISO/IEC/IEEE 29148:2018. Systems and Software Engineering — Life Cycle Processes — Requirements Engineering. https://www.iso.org/standard/72089.html

7. ISO/IEC 12207:2017. Systems and Software Engineering — Software Life Cycle Processes. https://www.iso.org/standard/63712.html

8. ISO/IEC 25010:2023. Systems and Software Quality Models. https://www.iso.org

9. IEEE Standards Association. (2024). Requirements Engineering Standards. https://standards.ieee.org

10. International Institute of Business Analysis. (2024). *BABOK Guide v3*. https://www.iiba.org

11. Project Management Institute. (2021). *PMBOK Guide* (7th ed.). https://www.pmi.org

12. Atlassian. (2025). Requirements Management Guide. https://www.atlassian.com/agile/project-management/requirements

13. Atlassian Jira Software Documentation. (2025). https://support.atlassian.com/jira-software-cloud

14. IBM Engineering Requirements Management DOORS Next Documentation. (2025). https://www.ibm.com/docs/en/engineering-lifecycle-management-suite

15. Microsoft Azure DevOps Documentation. (2025). https://learn.microsoft.com/en-us/azure/devops

16. Azure Boards Documentation. (2025). https://learn.microsoft.com/en-us/azure/devops/boards

17. Jama Software. (2025). Requirements Management Best Practices. https://www.jamasoftware.com

18. Perforce Helix ALM Documentation. (2025). https://www.perforce.com/products/helix-alm

19. Siemens Polarion ALM Documentation. (2025). https://www.polarion.com

20. Lucidchart. (2025). Requirements Gathering Techniques. https://www.lucidchart.com

21. Object Management Group. (2024). Unified Modeling Language (UML). https://www.omg.org/spec/UML

22. Visual Paradigm. (2025). Requirements Engineering Guide. https://www.visual-paradigm.com

23. Cohn, M. (2004). *User Stories Applied: For Agile Software Development*. Addison-Wesley. https://www.informit.com

24. Leffingwell, D. (2023). *SAFe Requirements Model*. https://scaledagileframework.com

25. Agile Alliance. (2025). Agile Requirements Practices. https://www.agilealliance.org

26. Scrum.org. (2025). Product Backlog Management. https://www.scrum.org

27. IEEE Software Magazine. (2024). Modern Trends in Requirements Engineering. https://www.computer.org/csdl/magazine/so

28. Oracle. (2025). Application Lifecycle Management Documentation. https://docs.oracle.com

29. Hassan, A., & Mahmood, S. (2023). Requirements Traceability in Software Projects. *Journal of Systems and Software*, 198, 111567. https://doi.org/10.1016/j.jss.2023.111567

30. Kumar, R., & Singh, P. (2024). Automated Requirements Analysis Using Machine Learning. *Software Quality Journal*, 32(2), 521–540. https://doi.org/10.1007/s11219-023-09658-1

31. Zhang, L., & Wang, Y. (2023). Managing Requirements Changes in Agile Development. *Information and Software Technology*, 158, 107171. https://doi.org/10.1016/j.infsof.2023.107171

32. Chen, J., & Liu, H. (2024). Comparative Analysis of Requirements Management Tools. *Journal of Software Engineering Research and Development*, 12(1), 14–29. https://doi.org/10.1186/s40411-024-00231-8

33. IEEE Computer Society. (2025). Guide to Software Requirements Specification. https://www.computer.org

34. Oracle Academy. (2024). Software Development Lifecycle and Requirements. https://academy.oracle.com

35. Gartner. (2024). Market Guide for Requirements Management Tools. https://www.gartner.com

36. TechTarget. (2024). Requirements Management Definition and Best Practices. https://www.techtarget.com/searchsoftwarequality

37. Red Hat. (2025). Agile Development and Requirements Management. https://www.redhat.com

38. OpenText. (2025). Requirements and Quality Management Solutions. https://www.opentext.com

39. Berenbach, B., Paulish, D., Kazmeier, J., & Rudorfer, A. (2009). *Software & Systems Requirements Engineering*. McGraw-Hill. https://www.mheducation.com

40. Wieringa, R. (2020). *Requirements Engineering: Frameworks for Understanding*. Springer. https://link.springer.com/book/10.1007/978-3-030-53913-1

БІБЛІОГРАФІЧНА ДОВІДКА

Тема бакалаврської роботи: "Методи управління вимогами до програмного забезпечення"

Обсяг пояснювальної записки: 78 аркушів

Дата закінчення бакалаврської роботи 10 червня 2026р.

Підпис студента _____