МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В. Н. Каразіна

Факультет: **ННІ Каразінський банківський інститут**
Кафедра: **Інформаційних технологій та математичного моделювання**
Спеціальність: **122 Комп'ютерні науки**
Освітня програма: **Комп'ютерні науки**

Група: АК-21М **денна форма навчання**

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«ДОСЛІДЖЕННЯ ВПЛИВУ ГНУЧКИХ МЕТОДОЛОГІЙ НА ЕФЕКТИВНІСТЬ ПРОЦЕСУ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ»

ЗА НАКАЗОМ № 4601-5/3262 ВІД 15 ВЕРЕСНЯ 2025 РОКУ

здобувача вищої освіти **Кулалаєва Нікіти Андрійовича**

Робота допущена до захисту в ЕК
протокол кафедри ІТММ № 5 від 02.12.2025 р.

В.о. завідувача кафедри ІТММ
PhD
_____**Ковальчук Д. М.**

Науковий керівник
PhD з спеціальності комп'ютерні науки
_____**Ковальчук Д. М.**

м. Харків 2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет імені В. Н. Каразіна

Факультет навчально-науковий інститут “Каразінський банківський інститут”
Кафедра інформаційних технологій та математичного моделювання
Рівень вищої освіти другий (магістерський)
Спеціальність 122 Комп’ютерні науки
Освітня програма Комп’ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри

Н. І. Стяглик

“15” вересня 2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ (ПРОЕКТ)

Кулалаєва Нікіти Андрійовича
(прізвище, ім’я, по батькові студента)

1. Тема роботи «Дослідження впливу гнучких методологій на ефективність процесу розробки програмного забезпечення»

Керівник роботи PhD з спеціальності комп’ютерні науки Ковальчук Д.М.

затверджені наказом по університету від “15” вересня 2025 року №4601-5/3262

2. Строк подання студентом роботи 24 листовада 2025 р.

3. Перелік питань, які потрібно розробити:
У розділі 1: Провести аналіз класичних та гнучких моделей життєвого циклу розробки програмного забезпечення і визначити їхні переваги та недоліки.
У розділі 2: Дослідити вплив гнучких методологій на ефективність управління проєктами.
У розділі 3: Розробити програмну систему управління проєктами на основі гнучких методологій (Agile).

4. План роботи

№ з/п	Назви етапів роботи
1	Вибір здобувачем теми кваліфікаційної магістерської роботи
2	Затвердження плану і завдання кваліфікаційної магістерської роботи
3	Здача кваліфікаційної магістерської роботи керівнику
4	Підпис кваліфікаційної магістерської роботи у керівника
5	Підпис кваліфікаційної магістерської роботи у нормо контролера
6	Допуск завідувачем кафедри до захисту кваліфікаційної магістерської роботи
7	Захист кваліфікаційної магістерської роботи

5. Дата видачі завдання 15 вересня 2025 року

Студент _____ Н. А. Кулалаєв
Підпис ініціали, прізвище

Керівник роботи _____ Д. М. Ковальчук
підпис ініціали, прізвище

РЕФЕРАТ
НА КВАЛІФІКАЦІЙНУ МАГІСТЕРСЬКУ РОБОТУ
«ДОСЛІДЖЕННЯ ВПЛИВУ ГНУЧКИХ МЕТОДОЛОГІЙ НА ЕФЕКТИВНІСТЬ ПРОЦЕСУ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ»
Кулалаєв Нікіта Андрійович

Кваліфікаційна бакалаврська робота містить 86 сторінок, 25 рисунків, 2 таблиці, список використаних джерел із 38 найменувань.

Об'єктом дослідження є процес розроблення програмного забезпечення в межах проєктного середовища.

Предметом дослідження є методологічні та програмні засоби, що впливають на ефективність розробки програмного забезпечення на основі гнучких підходів.

Мета кваліфікаційної магістерської роботи полягає у дослідженні впливу гнучких методологій на ефективність процесу розроблення програмного забезпечення та розробленні інформаційної системи керування проєктами, яка реалізує принципи Agile і містить засоби побудови діаграм Ганта для аналізу виконання завдань та визначення критичного шляху.

Завданнями кваліфікаційної магістерської роботи є:

– провести аналіз класичних і гнучких методологій розроблення програмного забезпечення;

– здійснити порівняльну характеристику основних підходів до керування проєктами;

– дослідити особливості процесу планування в умовах гнучких підходів і визначити критерії ефективності управління;

– розробити програмне забезпечення для системи керування програмними проєктами з підтримкою побудови діаграм Ганта;

– реалізувати алгоритм визначення критичного шляху для оцінювання тривалості проєкту та ефективності використання ресурсів.

Актуальність дослідження. Стрімкий розвиток інформаційних технологій, зростання вимог до якості, швидкості та гнучкості процесів створення програмного забезпечення зумовлюють необхідність удосконалення підходів до розроблення. У сучасних IT-командах важливо забезпечити прозорість, передбачуваність і контроль за виконанням завдань, що досягається за допомогою інтеграції методів планування та візуалізації проєктних процесів у рамках гнучких методологій, зокрема за допомогою діаграм Ганта.

За результатами дослідження: було розроблено вебсистему керування проєктами, яка реалізує принципи Agile та дозволяє автоматизовано будувати діаграми Ганта, оновлювати їх у реальному часі та визначати критичний шлях виконання завдань. Система забезпечує покращену координацію роботи команди, своєчасне виявлення затримок та оптимізацію використання ресурсів.

Практична новизна. Розроблено інформаційну систему керування проєктами на основі сучасного стеку вебтехнологій React, Node.js та MySQL, що підтримує модулі авторизації та реєстрації користувачів, управління командами, обмін повідомленнями, ведення завдань та побудову діаграм Ганта з визначенням критичного шляху.

Одержані результати можуть бути використані для підвищення ефективності командної роботи в IT-компаніях, планування та контролю виконання проєктів, а також як навчальний і демонстраційний матеріал для впровадження Agile-підходів у процес розроблення програмного забезпечення.

КЛЮЧОВІ СЛОВА: УПРАВЛІННЯ ПРОЄКТАМИ, ГНУЧКІ МЕТОДОЛОГІЇ, AGILE, ДІАГРАМА ГАНТА, КРИТИЧНИЙ ШЛЯХ, REACT, NODE.JS, MYSQL, КОМАНДНА ВЗАЄМОДІЯ, ВЕБСИСТЕМА.

ABSTRACT
AT QUALIFICATION MASTER'S WORK
«RESEARCH ON THE IMPACT OF AGILE METHODOLOGIES ON THE EFFICIENCY OF SOFTWARE DEVELOPMENT PROCESSES»
Nikita Kulalaiev

The qualification bachelor's thesis contains 86 pages, 25 figures, 2 tables, and a list of 38 references.

The object of the study is the software development process within a project-based environment.

The subject of the study is the methodological and software tools that affect the efficiency of software development based on agile approaches.

The aim of the qualification bachelor's thesis is to investigate the impact of agile methodologies on the efficiency of the software development process and to develop an information system for project management that implements Agile principles and includes tools for creating Gantt charts to analyze task execution and determine the critical path.

The tasks of the qualification bachelor's thesis are:

– to analyze classical and agile software development methodologies;

– to carry out a comparative analysis of the main project management approaches;

– to study the features of the planning process under agile approaches and determine criteria for management efficiency;

– to develop software for a project management system with support for building Gantt charts;

– to implement an algorithm for determining the critical path to evaluate project duration and resource utilization efficiency.

Relevance of the study. The rapid development of information technologies, increasing requirements for quality, speed, and flexibility of software development processes necessitate the improvement of development approaches. In modern IT teams, it is important to ensure transparency, predictability, and control over task execution, which is achieved through the integration of planning methods and visualization of project processes within agile methodologies, particularly using Gantt charts.

Research results: a web-based project management system was developed that implements Agile principles and allows automated construction of Gantt charts, real-time updates, and determination of the critical path for task execution. The system improves team coordination, timely identification of delays, and optimization of resource usage.

Practical novelty. An information system for project management was developed using the modern web technology stack React, Node.js, and MySQL, supporting modules for user authorization and registration, team management, messaging, task tracking, and Gantt chart creation with critical path determination.

The obtained results can be used to improve team efficiency in IT

companies, plan and control project execution, as well as serve as educational and demonstration material for implementing Agile approaches in the software development process.

KEYWORDS: PROJECT MANAGEMENT, AGILE METHODOLOGIES, AGILE, GANTT CHART, CRITICAL PATH, REACT, NODE.JS, MYSQL, TEAM COLLABORATION, WEB SYSTEM.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧОК, СИМВОЛІВ І ТЕРМІНІВ

API – application programming interface (програмний інтерфейс застосунків);

RUP – Rational Unified Process (раціональний уніфікований процес);

RAD – Rapid Application Development (швидка розробка застосунків);

XP – Extreme Programming (екстремальне програмування);

UI – User Interface (користувацький інтерфейс);

UX – User Experience (досвід користувача);

БД – база даних;

ЖЦ – життєвий цикл;

ПЗ – програмне забезпечення;

СУБД – система управління базами даних;

ВСТУП

Стрімкий розвиток інформаційних технологій, зростання вимог до якості, швидкості та гнучкості процесів створення програмного забезпечення зумовлюють необхідність постійного вдосконалення підходів до його розроблення. У сучасних умовах бізнесу, де ринкові вимоги змінюються надзвичайно швидко, традиційні каскадні моделі життєвого циклу виявляються малоефективними через свою жорстку послідовність етапів та обмежену здатність реагувати на зміни. Саме тому дедалі більшого поширення набувають гнучкі методології (Agile), які орієнтовані на адаптивність, командну взаємодію, ітеративність та постійне вдосконалення продукту.

Застосування гнучких підходів, таких як Scrum, Kanban, Extreme Programming, Rapid Application Development чи Rational Unified Process, дозволяє ефективно організовувати роботу команд, підвищувати продуктивність та знижувати ризики невідповідності кінцевого продукту вимогам замовника. Водночас упровадження Agile-практик потребує чіткого розуміння їхніх принципів, оцінки доцільності застосування для різних типів проєктів і аналізу впливу на загальну ефективність процесу розроблення програмного забезпечення.

Актуальність теми дослідження зумовлена необхідністю інтеграції методів планування та візуалізації проєктних процесів у рамках гнучких методологій, що забезпечують прозорість, передбачуваність і контроль за виконанням завдань у команді. Одним із ключових інструментів управління у таких системах є діаграма Ганта, яка дозволяє візуально представити часові залежності між завданнями, визначити критичний шлях, своєчасно виявити затримки та оптимізувати розподіл ресурсів. Створення інформаційної системи, здатної автоматизовано будувати діаграми Ганта, оновлювати їх у реальному часі та визначати критичний шлях, є актуальним завданням для сучасних проєктно-орієнтованих ІТ-команд.

Метою роботи є дослідження впливу гнучких методологій на ефективність процесу розроблення програмного забезпечення та розроблення інформаційної системи керування проєктами, яка реалізує принципи Agile і містить засоби побудови діаграм Ганта для аналізу виконання завдань та визначення критичного шляху.

Для досягнення поставленої мети необхідно виконати такі завдання:

– провести аналіз класичних і гнучких методологій розроблення програмного забезпечення;

– здійснити порівняльну характеристику основних підходів до керування проєктами;

– дослідити особливості процесу планування в умовах гнучких підходів і визначити критерії ефективності управління;

– розробити програмне забезпечення для системи керування програмними проєктами з підтримкою побудови діаграм Ганта;

– реалізувати алгоритм визначення критичного шляху для оцінювання тривалості проєкту та ефективності використання ресурсів.

Об’єктом дослідження є процес розроблення програмного забезпечення в межах проєктного середовища.

Предметом дослідження є методологічні та програмні засоби, що впливають на ефективність розробки програмного забезпечення на основі гнучких підходів.

У першому розділі здійснено аналіз основних моделей життєвого циклу програмного забезпечення, розглянуто класичні та гнучкі методології, визначено їхні особливості, переваги та недоліки.

У другому розділі досліджено вплив гнучких підходів на ефективність управління програмними проєктами, проаналізовано процес планування, розподілу ролей, побудови календарних графіків і використання діаграм Ганта для контролю виконання робіт.

У третьому розділі розроблено програмну реалізацію системи керування проєктами, що базується на принципах Agile. Описано архітектуру

системи, діаграми прецедентів, модель бази даних та програмні компоненти, включно з інтерфейсами авторизації, управління командами, обміну повідомленнями, ведення завдань і побудови діаграм Ганта з підтримкою визначення критичного шляху.

Результати дослідження мають практичну цінність для ІТ-компаній, менеджерів проєктів і команд розробників, оскільки дозволяють підвищити ефективність планування, покращити координацію між учасниками, скоротити час реалізації проєктів і своєчасно виявляти ризики, пов’язані із затримками у виконанні завдань.

РОЗДІЛ 1
АНАЛІЗ МОДЕЛЕЙ ЖИТТЄВОГО ЦИКЛУ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1. Стратегії розробки програмного забезпечення

Процес створення програмного забезпечення протягом останніх десятиліть зазнав суттєвих трансформацій, що пов’язано як з розвитком технологій, так і з накопиченням практичного досвіду в індустрії. На ранніх етапах становлення обчислювальної техніки процес розроблення ПЗ будувався за надзвичайно простим принципом: програмісти створювали вихідний код, а потім неодноразово виправляли помилки, що виникали під час роботи продукту (рис. 1.1). Такий примітивний підхід виправдовував себе в умовах низької складності систем, невеликої кількості користувачів і мінімальних вимог до функцій та надійності [1, 2]. Проте з ускладненням програмних систем, збільшенням кількості користувачів та зростанням ролі програмного забезпечення в бізнес-процесах виникла необхідність у систематизації процесу розроблення та появі спеціалізованих методологій, здатних забезпечити більш передбачуваний, керований та ефективний життєвий цикл програмного продукту [3].

Рис. 1.1. Рання модель розроблення програмного забезпечення

Вибір стратегії розробки завжди зумовлений характером проєкту, особливостями команди, рівнем залучення користувачів, вимогами до кінцевого продукту та зовнішніми факторами, такими як часові та бюджетні

обмеження. Неправильно обрана методологія може призвести до зростання витрат, затримок у розробці, зниження якості кінцевого продукту або до невідповідності рішення очікуванням замовника. Водночас вдалий вибір підходу дає змогу оптимізувати роботу команди, підвищити продуктивність і забезпечити відповідність результату поставленим вимогам [4].

Однією з перших формалізованих стратегій стала каскадна модель, яка передбачає послідовний, одноразовий прохід усіх фаз життєвого циклу: від аналізу вимог до експлуатації. Цей підхід базується на принципі, що перед початком розроблення повинні бути чітко визначені всі функціональні та нефункціональні вимоги. Кожен етап виконується лише після завершення попереднього, а повернення до попередніх фаз вважається небажаним або надзвичайно складним. Такий підхід вважається структурованим, дозволяє чітко планувати етапи роботи, розподіляти ресурси та контролювати виконання. Він особливо ефективний у проєктах з передбачуваними, стабільними вимогами та низьким рівнем невизначеності. Наприклад, він добре підходить для проєктів, що реалізуються у сфері оборонних систем, систем керування або фінансових платформ із чітко регламентованими функціями [5].

Попри очевидні переваги, каскадна модель має низку критичних недоліків. Найбільшим з них є низька гнучкість – неможливість вносити зміни після того, як вимоги вже зафіксовані. В умовах сучасного ринку, де ПЗ часто створюється в умовах невизначеності й швидкої зміни потреб користувачів, цей фактор стає вирішальним. Крім того, тестування відбувається лише на завершальних етапах, що ускладнює виявлення та виправлення критичних проблем, які можуть вплинути на архітектуру та основні функції продукту. Також замовник фактично не бачить продукт до його завершення, що підвищує ризик невідповідності очікуванням [6].

Зі зростанням масштабів проєктів і необхідності швидше реагувати на зміни бізнес-вимог виникла інкрементна стратегія. На відміну від каскадного підходу, вона передбачає розроблення системи частинами – невеликими

функціональними блоками, кожен з яких створюється в окремому циклі. Це дозволяє отримувати робочий продукт уже після перших етапів, а також забезпечувати постійний зворотний зв'язок від користувачів. Кожна нова версія програми містить розширену функціональність, але при цьому спирається на ядро, створене раніше. Таким чином забезпечується поступове формування повноцінного продукту [5-7].

Інкрементний підхід сприяє зниженню ризиків, оскільки помилки можна виявляти на ранніх етапах, а зміни у вимогах легше інтегрувати в нові інкременти. Цей підхід часто поєднується з прототипуванням – створенням пробних версій системи, які допомагають краще зрозуміти вимоги користувачів і визначити напрям подальшої розробки. Проте інкрементність потребує ретельного управління, оскільки неправильний розподіл функціональності або відкладання складних задач на пізніші етапи може призвести до технічної заборгованості чи хаотичного розвитку системи [7].

Іншим важливим підходом, що став логічним розвитком інкрементності, є еволюційна стратегія. Вона передбачає поступове формування вимог і реалізацію системи в кількох ітераціях, кожна з яких завершується створенням працездатної версії продукту. Відмінністю еволюційного підходу є більший акцент на поступове уточнення вимог, що робить його особливо корисним у проєктах із високим рівнем невизначеності. У багатьох випадках початкові вимоги можуть бути нечіткими або неповними, а процес їх уточнення відбувається паралельно із розробленням. Прототипи відіграють тут важливу роль, оскільки дозволяють користувачам оцінити базову функціональність і надати зворотний зв’язок.

Еволюційний підхід забезпечує високу гнучкість і дає можливість реагувати на зміни на будь-якому етапі проєкту. Він також дозволяє замовнику брати активну участь у розробці, що знижує ризик розбіжностей між очікуваннями та результатом. Проте недоліком є складність планування, оскільки заздалегідь передбачити кількість ітерацій та їхню тривалість часто неможливо. Крім того, команді необхідні потужні інструменти та висока

кваліфікація в управлінні процесом, що збільшує вимоги до рівня зрілості організації [7, 8].

У сучасних умовах ринок програмного забезпечення вимагає максимальної адаптивності та можливості швидко виводити продукти на ринок. Саме тому дедалі більшої популярності набувають гнучкі методології, що базуються на принципах еволюційної розробки, але підсилені практиками, орієнтованими на постійне вдосконалення, тісну взаємодію з клієнтом і самокерованість команд. Agile-підходи стали відповіддю на обмеження традиційних моделей, особливо у сфері інноваційних продуктів і стартапів, де попереднє детальне планування часто неможливе або недоцільне. Вони дозволяють адаптуватися до ринку, отримувати швидкий зворотний зв’язок та оптимізувати процеси відповідно до змін вимог і бізнес-пріоритетів.

Таким чином, в історії розвитку стратегій створення програмного забезпечення простежується чітка еволюція: від жорстких, лінійних моделей до гнучких, ітеративних підходів. Традиційні моделі залишаються актуальними там, де вимоги майже не змінюються, а процеси мають високий рівень регламентації. Водночас проєкти з високим рівнем динаміки потребують гнучких стратегій, здатних забезпечити адаптивність та швидкий зворотний зв’язок. Розуміння особливостей кожного підходу дозволяє обирати оптимальну методику для конкретних умов і забезпечувати максимальну ефективність процесу розробки.

1.2. Класичні моделі розробки програмного забезпечення

1.2.1. Водоспадна модель

Однією з найвідоміших та історично перших формалізованих методологій створення програмного забезпечення є водоспадна (каскадна) модель. Її поява стала результатом прагнення надати процесу розроблення

ПЗ чіткої логічної структури та забезпечити контроль над кожним етапом життєвого циклу продукту. Водоспадна модель базується на принципі суворої послідовності виконання робіт: завершення поточного етапу є передумовою для початку наступного (рис. 1.2). Такий порядок створює ефект «водоспаду», коли потік робіт рухається зверху вниз, переходячи від визначення вимог до експлуатації та супроводу [1, 8].

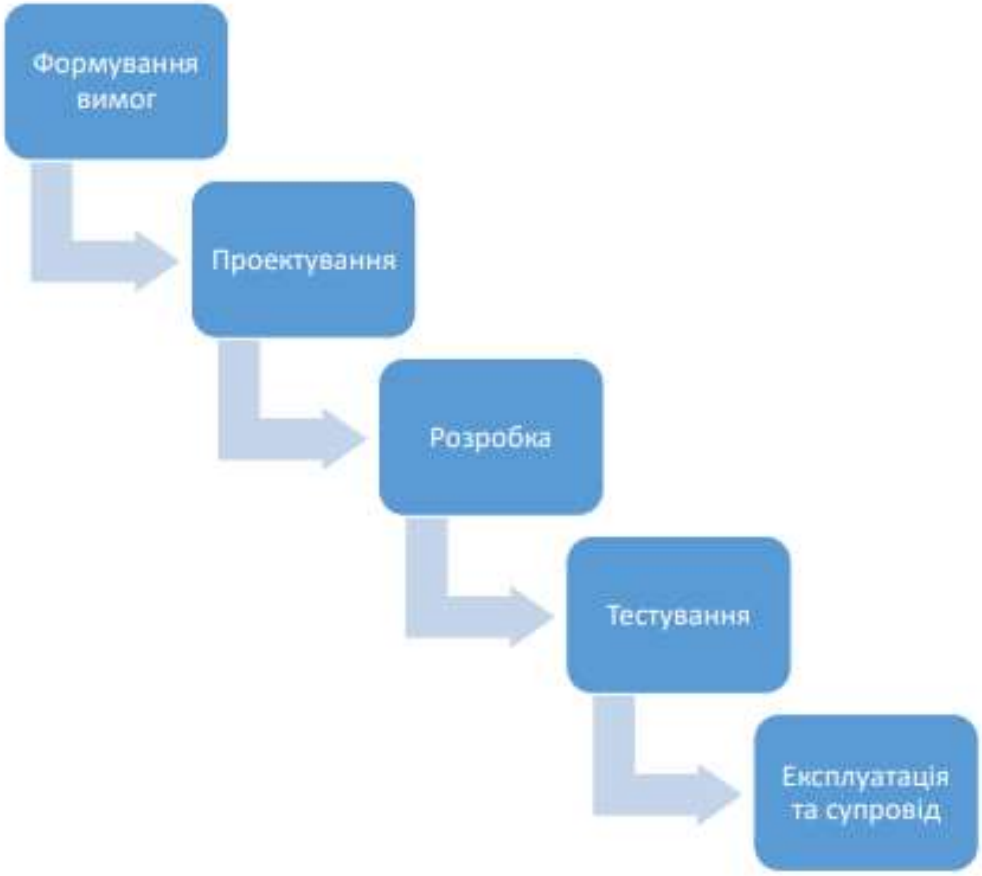

Рис. 1.2. Водоспадна модель розробки програмного забезпечення

У класичному поданні водоспадна модель передбачає послідовне проходження таких фаз: формування вимог, проектування, розроблення (кодинг), тестування, введення в експлуатацію та подальший супровід програмного продукту. Центральною ідеєю є те, що до переходу на новий етап повинно бути завершено всі роботи на попередньому, а також створено необхідний комплект документації, який слугує основою для подальшої

діяльності. Саме документованість і чіткість переходів між етапами вважаються ключовими характеристиками даного підходу.

Етап формування вимог має принципове значення, оскільки саме на цьому етапі визначаються всі функціональні та нефункціональні характеристики майбутньої системи. Передбачається, що вимоги можуть і повинні бути сформульовані повністю ще до початку розроблення, а будь-які зміни в подальшому небажані. Далі здійснюється проектування системи – формується архітектура програмного забезпечення, визначається структура його компонентів, алгоритми їх взаємодії, моделі даних та інтерфейси. Результатом етапу є повний комплект проектної документації, що визначає технічні основи майбутньої реалізації [1, 8].

На етапі розроблення програмісти виконують безпосереднє кодування, тобто трансформують проектні рішення у програмний код. Усі компоненти, визначені під час проектування, створюються, інтегруються та готуються до перевірки. Після завершення розроблення починається тестування, основна мета якого – виявлення помилок, перевірка відповідності системи вимогам і забезпечення необхідного рівня якості перед передачею продукту в експлуатацію. Лише після завершення тестування система може бути прийнята замовником та введена в експлуатацію. Фінальним етапом є супровід, який передбачає усунення помилок, адаптацію системи до змін у зовнішньому середовищі, а іноді – модернізацію відповідно до нових потреб користувачів [1, 8].

Попри критичну послідовність, модель допускає обмежену паралельність робіт. Наприклад, підготовка тестової документації може розпочинатися наприкінці стадії проектування, а не після завершення розроблення коду. Також можливе часткове накладання окремих процесів, якщо вони не залежать жорстко один від одного. Проте навіть у таких випадках логіка «завершили – переходимо далі» зберігається, і команди, що виконують різні етапи, як правило, працюють незалежно [9].

Перевагами водоспадної моделі є передбачуваність, контрольованість і висока структурованість процесу. Завдяки докладному плануванню і документуванню замовник отримує чітке уявлення про бюджет, терміни та обсяг робіт. Проєкт легко контролювати, а відповідальність команд чітко розмежована. Такий підхід особливо ефективний у випадках, коли вимоги до продукту є стабільними, зрозумілими та не змінюються протягом усього життєвого циклу проєкту. Саме тому каскадна модель традиційно застосовувалася у великих інженерних системах, оборонних проєктах, а також у розробці критичного програмного забезпечення, де регламентовані процеси відіграють ключову роль [9].

Однак строгість каскадного підходу є водночас його найбільшим обмеженням. У реальних умовах замовники часто не здатні сформулювати всі вимоги на початковій стадії, а зміни потреб або зовнішніх факторів можуть виникати на будь-якому етапі. Водоспадна модель практично не допускає внесення змін у вимоги після їх фіксації, що призводить до високих ризиків невідповідності кінцевого продукту потребам користувачів. Іншим суттєвим недоліком є відкладений зворотний зв’язок: замовник бачить робочий продукт лише наприкінці розроблення, коли виправлення серйозних недоліків стає складним, дорогим або навіть технічно неможливим.

Крім того, виявлення критичних помилок у пізній фазі тестування може вимагати повернення до етапів проектування або навіть формулювання вимог, що руйнує саму концепцію суворої послідовності. Для великих і тривалих проєктів накопичення технічної заборгованості або неправильна інтерпретація вимог може привести до значних фінансових та часових втрат.

Таким чином, водоспадна модель є фундаментальним етапом розвитку методологій програмної інженерії та залишається актуальною у сферах, де проєкти мають чіткі вимоги, високий рівень формалізації та низьку ймовірність змін. Проте в умовах динамічного ринку та швидкого розвитку технологій її застосування обмежене, що зумовило появу і популяризацію ітеративних та гнучких методів розроблення ПЗ [8-10].

1.2.2. V-подібна модель

V-подібна модель життєвого циклу програмного забезпечення є еволюційним розвитком класичної каскадної (Waterfall) моделі. Її основна ідея полягає у встановленні прямого зв'язку між кожним етапом проєктування та відповідним етапом тестування, що дозволяє рано виявляти та попереджувати помилки у вимогах та архітектурних рішеннях. На відміну від класичної каскадної моделі, де тестування починається лише після завершення реалізації системи, у V-подібній моделі планування тестування здійснюється вже на стадії формування вимог [1, 9].

Структурно модель представлена у вигляді літери V, де ліва сторона відображає послідовний процес аналізу та проєктування, а права – відповідні рівні тестування та валідації (рис. 1.3). Центральна точка «V» – це етап безпосередньої розробки програмного продукту.

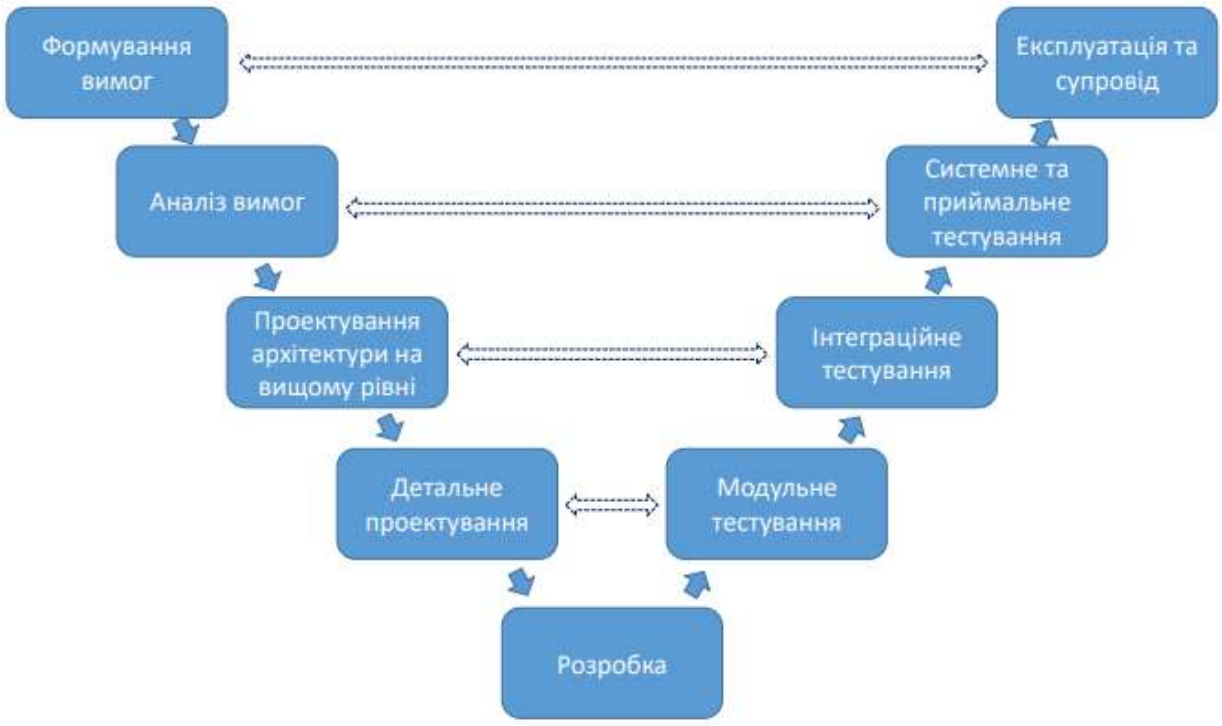

Рис. 1.3. V-подібна модель розробки програмного забезпечення

Основні етапи V-подібної моделі включають [10]:

– формування вимог – визначення функціональних та нефункціональних вимог до системи, формування специфікацій; одночасно розпочинається розробка загальної стратегії тестування та критеріїв приймання;

– аналіз вимог – деталізація та документування функціональних вимог до програмної частини системи, підготовка сценаріїв майбутнього приймального тестування;

– проєктування архітектури на високому рівні – визначення структурних компонентів системи та їхніх взаємозв’язків; формування критеріїв інтеграційного тестування;

– детальне проєктування – розробка модульної структури, алгоритмів та інтерфейсів; створення планів модульного тестування;

– розроблення – написання коду та інтеграція модулів згідно з проєктною документацією;

– модульне тестування – перевірка окремих компонентів програмного забезпечення на відповідність розробленим технічним специфікаціям;

– інтеграційне тестування – перевірка коректності взаємодії модулів, усунення помилок інтеграції;

– системне тестування – комплексна перевірка функціональності всієї системи відповідно до технічних вимог;

– приймальне тестування – підтвердження відповідності системи вимогам замовника, підготовка до передачі у промислову експлуатацію;

– експлуатація та супровід – внесення змін, адаптація системи, усунення виявлених у процесі використання помилок.

Ключовою перевагою V-подібної моделі є раннє планування процесів тестування та чітке відображення взаємозв’язку етапів проєктування й тестування, що забезпечує структурованість і підвищує якість кінцевого продукту. Саме тому модель часто застосовується у великих та критично важливих проєктах (наприклад, у медичних інформаційних системах,

авіоніці, банківських системах), де надійність та прогнозованість процесів мають першочергове значення [8-10].

Проте, подібно до каскадної моделі, V-подібний підхід має обмеження. Він передбачає стабільність вимог і допускає мінімальні зміни під час розроблення. У випадку необхідності внесення суттєвих змін на пізніх етапах витрати на їх реалізацію значно зростають. Крім того, модель потребує ретельної та якісної початкової документації, а наочний результат замовник отримує лише після завершення основних етапів розроблення.

Отже, V-подібна модель забезпечує високу структурованість процесу розроблення, сприяє зменшенню помилок проєктування та контролю якості на кожному етапі, проте потребує стабільності вимог і чітко визначеної сфери застосування.

1.2.3. Інкрементна модель

Інкрементна модель є однією з класичних моделей життєвого циклу програмного забезпечення, що реалізує інкрементну стратегію розроблення. В її основу покладено поетапне нарощування функціональності програмного продукту шляхом створення та послідовного впровадження окремих інкрементів. Кожен інкремент являє собою завершену частину системи, яка додає нові можливості до вже наявного функціоналу (рис. 1.4) [1, 11].

Перший інкремент зазвичай включає базову функціональність програмного продукту, наприклад, інтерфейс користувача або базові модулі аутентифікації. Кожен наступний інкремент розширює систему новими компонентами, функціями або сервісами, забезпечуючи поступове наближення ПЗ до кінцевої мети проєкту. Таким чином, після завершення кожного циклу розроблення замовнику може бути представлена робоча версія продукту з певним набором функцій, що дозволяє отримувати цінність від програмного забезпечення ще до повного завершення проекту [11, 12].

Рис. 1.4. Інкрементна модель розробки програмного забезпечення

Досить часто інкрементна модель поєднується з ітераційним підходом, формуючи ітераційно-інкрементну модель (IID, Iterative and Incremental Development). У цьому випадку система розробляється невеликими частинами, а кожна частина проходить всі основні етапи життєвого циклу – від формування вимог і проєктування до реалізації та тестування. Кожна ітерація є самостійно завершеною одиницею розробки, що дозволяє уточнювати вимоги та коригувати напрям проєкту в процесі роботи.

Постійне уточнення вимог і можливість зупинити ітерації у будь-який момент, коли досягнуто задовільний результат, забезпечує гнучкість моделі та дає змогу зменшити ризики помилок стратегічного планування. Важливим результатом використання ітераційно-інкрементної моделі є можливість випуску мінімально життєздатного продукту (MVP, Minimum Viable Product), що дозволяє тестувати продукт з реальними користувачами на ранніх стадіях.

Серед ключових переваг інкрементної та ітераційно-інкрементної моделей виділяють [7, 10]:

– зниження ризиків шляхом раннього виявлення конфліктів між вимогами та реалізацією;

– можливість динамічного формування та адаптації вимог;

– гнучке управління пріоритетами функціональності;

– регулярний контакт із замовником та отримання зворотного зв’язку;

– швидке створення базової робочої версії продукту (MVP).

Водночас зазначена модель має і певні недоліки [12]:

– складність у забезпеченні стабільної та масштабованої архітектури без наявності чіткого глобального плану;

– відсутність фіксованих термінів і бюджету на початкових етапах;

– необхідність постійної участі замовника та кінцевих користувачів у процесі розробки, що не завжди можливо організувати на практиці.

Таким чином, інкрементна та ітераційно-інкрементна моделі поєднують у собі гнучкість, адаптивність і високу швидкість отримання результатів, а їх використання є ефективним у проєктах, де вимоги можуть змінюватися, а також у випадках, коли важливим є раннє отримання робочих версій програмного продукту.

1.2.4. Спіральна модель

Спіральна модель належить до моделей еволюційної стратегії життєвого циклу програмного продукту. Усі етапи ЖЦ у межах цієї моделі подаються у вигляді послідовних витків спіралі (рис. 1.5). На кожному витку виконуються основні процеси: аналіз і формалізація вимог, системне та детальне проєктування, розроблення, тестування, оцінювання результатів тощо. При цьому склад і глибина опрацювання процесів можуть змінюватися від витка до витка, але кожна ітерація обов’язково наближає проєкт до досягнення кінцевої мети [1, 11].

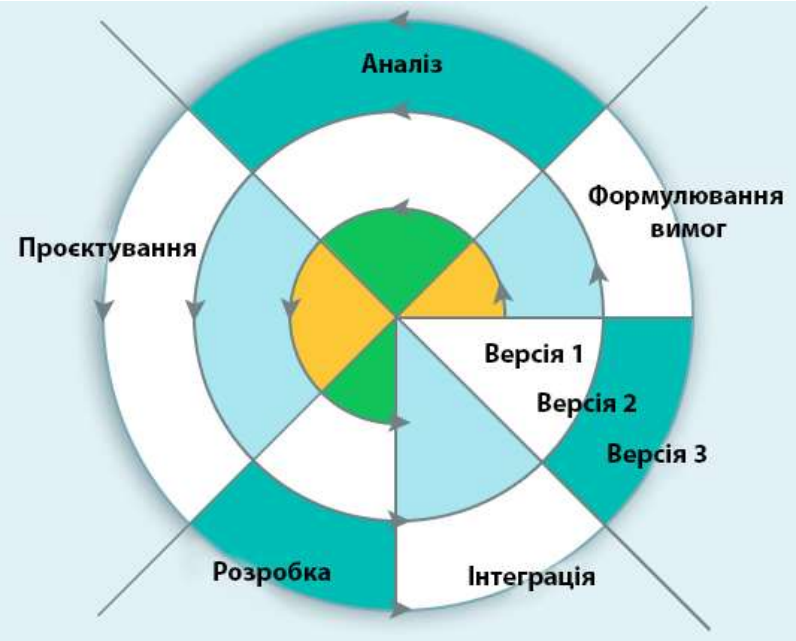

Рис. 1.5. Спіральна моделі життєвого циклу програмного забезпечення

Спіральна модель є характерною для проєктів, що передбачають використання інноваційних або нетипових технологій, коли замовник і розробник на початку не мають чіткого уявлення про остаточний вигляд продукту. Вимоги можуть бути нечітко визначені, а ризики – високими. У таких умовах розроблення здійснюється поетапно, з можливістю уточнення вимог або навіть відмови від подальшого розвитку проєкту за результатами проміжних оцінок.

Подальша розробка програмного продукту (ПП) може бути завершена як після упровадження готового продукту, так і значно раніше – наприклад, після первинного аналізу ризиків, якщо продовження робіт недоцільне.

Переваги спіральної моделі [12]:

– можливість швидкого випуску мінімально життєздатного продукту (MVP);

– гнучке формування та коригування вимог;

– підвищена гнучкість управління проєктом;

– отримання надійнішого та стійкішого ПЗ завдяки багаторазовому аналізу й усуненню недоліків на кожній ітерації;

– постійне вдосконалення процесу розроблення завдяки аналітичному підходу;

– зменшення ризиків замовника – можливість завершення нерентабельного проєкту з мінімальними втратами.

Недоліки спіральної моделі [12]:

– невизначеність у перспективах розвитку проєкту;

– збільшення тривалості та вартості розроблення;

– ризик тривалого «застрявання» на початкових етапах і нескінченного доопрацювання продукту;

– складність планування ресурсів та термінів через змінність вимог.

Щоб мінімізувати ризики, доцільно встановлювати часові обмеження на проходження кожного витка. Перехід до наступного етапу виконується відповідно до плану, навіть якщо частина запланованих робіт на поточній ітерації залишилася невиконаною. План формується на основі статистичних даних та досвіду керівника проєкту.

1.3. Гнучкі методології розробки програмного забезпечення

Гнучкі методології (Agile methodologies) – це група підходів до організації процесу розроблення програмного забезпечення, що орієнтовані на адаптивність, ітеративність, співпрацю із замовником та швидку доставку робочого програмного продукту. Agile виступає не лише методикою планування розроблення, а й філософією управління, що ґрунтується на цінностях самостійності команди, гнучкості та мінімізації бюрократії [13].

Agile-підхід є протилежністю традиційних (планово-орієнтованих) методологій, де основний акцент робиться на формальній документації, чіткій послідовності етапів та жорсткому дотриманні плану. Гнучкі методи розроблення натомість передбачають можливість внесення змін на будь-якому етапі проєкту, а також тісну взаємодію між замовником і командою розробників [13].

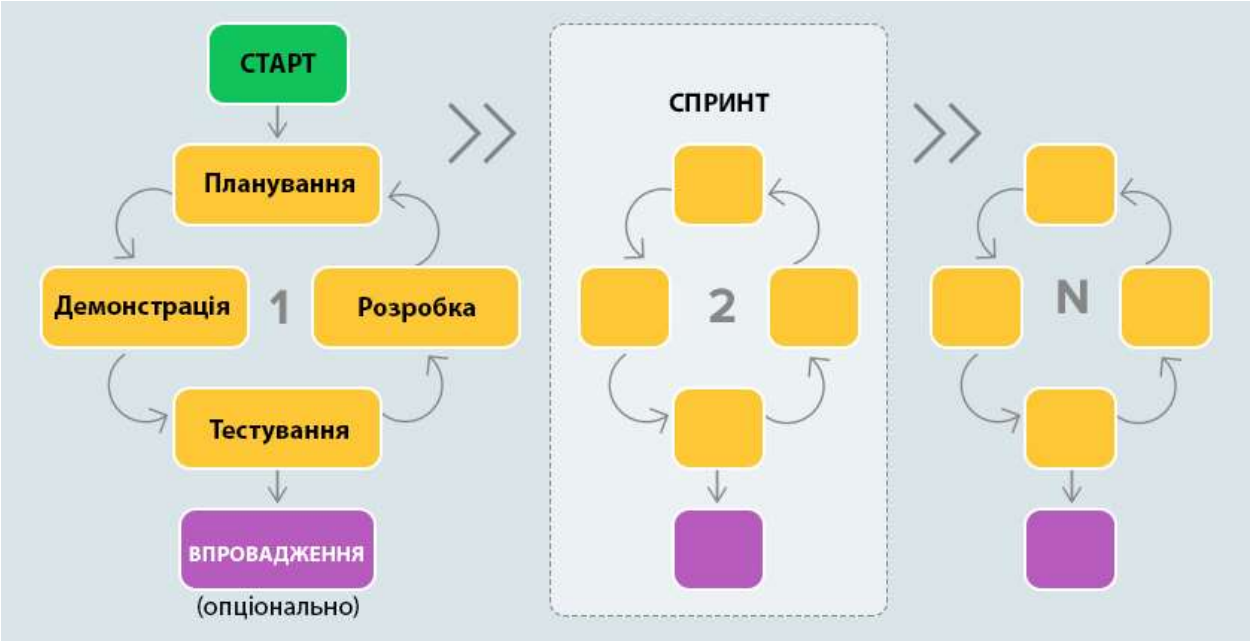

Рис. 1.6. Agile-підхід

Agile найчастіше застосовується в умовах високої мінливості вимог, динамічного ринку та необхідності швидкого реагування на зовнішні фактори. Це характерно для розроблення комерційних програмних продуктів, стартапів, сервісів електронної комерції, онлайн-платформ, мобільних застосунків тощо.

Основні цілі Agile-підходу [13-15]:

– скорочення часу виходу продукту на ринок (Time-to-Market);
– швидке створення мінімально життєздатного продукту (MVP);
– забезпечення постійного зворотного зв’язку із замовником;
– покращення якості за рахунок регулярного тестування та перевірки результатів;
– максимальна адаптивність до змін вимог;
– зниження ризиків, пов’язаних із невизначеністю на ранніх етапах.

Agile дозволяє уникнути ситуації, коли продукт, створений за формальними планами, на момент релізу вже не відповідає потребам користувача або ринку [13-15].

Відповідно до Маніфесту Agile, опублікованого у 2001 році, в основі методології лежать такі принципи [16]:

– люди та їх взаємодія важливіші, ніж процеси та інструменти;

– працюючий програмний продукт важливіший за вичерпну документацію;

– співпраця із замовником важливіша за формальні умови контракту;

– готовність до змін важливіша за дотримання початкового плану.

Це не означає, що процеси, документація, контракт або план не потрібні. Але в рамках Agile вони не є головними і не можуть блокувати роботу команди.

Принципи роботи Agile-команди [16]:

– активна взаємодія із замовником і користувачами продукту;

– автономність команди та висока відповідальність її учасників;

– короткі ітерації розроблення (зазвичай від одного дня до 2–4 тижнів);

– регулярне прийняття рішень на основі результатів роботи, а не припущень;

– постійне тестування та вдосконалення продукту;

– прозоре планування та щоденні комунікації (daily meetings).

Agile є найбільш ефективним, коли команда невелика (зазвичай до 7–10 осіб), а учасники – компетентні, мотивовані та здатні самостійно організовувати роботу.

Переваги Agile [16]:

– швидка реакція на зміни вимог;

– рання поява робочої версії продукту на ринку;

– мінімізація ризиків завдяки коротким циклам зворотного зв’язку;

– підвищення залученості замовника і як наслідок – вища відповідність очікуванням;

– покращення якості за рахунок частих перевірок та рефакторингу;

– постійний розвиток команди та вдосконалення процесів.

Недоліки Agile [16]:

– можливість хаотичності без досвідченого керівника або Scrum-майстра;

– труднощі з довгостроковим прогнозуванням загальних витрат і термінів;

– висока залежність від компетентності членів команди;

– необхідність високої включеності замовника;

– можливість нескінченних змін вимог.

Щоб уникнути ризику «нескінченного вдосконалення», у гнучких методологіях вводяться часові обмеження (тайм-бокси), а кожна ітерація має чіткий вимірний результат – інкремент продукту.

Agile є найефективнішим у середовищі, де:

– вимоги нестійкі і можуть змінюватися;

– проєкт має новаторський характер;

– неможливо передбачити всі ризики на старті;

– учасники зацікавлені у швидкому отриманні зворотного зв’язку.

Правило Agile-керування можна сформулювати таким чином: «Чим вищий рівень невизначеності – тим коротшою має бути ітерація». У деяких проєктах ітерації можуть становити 24 години або менше, що потребує щоденного аналізу виконаних завдань та планування наступних.

1.3.1. Методологія Scrum

Однією з найпоширеніших гнучких методологій управління проєктами є Scrum, який базується на ітеративному підході та постійному вдосконаленні процесу розроблення програмного забезпечення. У центрі методології Scrum знаходиться поняття «спринту» (sprint) – обмеженого в часі етапу, протягом якого команда виконує визначений обсяг завдань і створює інкремент продукту. Спринт зазвичай триває від одного до чотирьох тижнів, причому

тривалість обирається однакова для всього життєвого циклу проєкту, що забезпечує прогнозованість, регулярність керування та стабільність темпу роботи (рис. 1.7). Початок кожного спринту супроводжується процесом планування, під час якого команда спільно з власником продукту визначає цілі на найближчу ітерацію, аналізує вміст загального списку завдань – Product Backlog – та обирає ті елементи, що повинні бути реалізовані протягом поточного спринту. На основі вибраних задач формується Sprint Backlog, який, по суті, визначає робочий план команди на встановлений період. Обсяг завдань для спринту визначається з урахуванням складності робіт, наявних ресурсів і швидкості команди (velocity), яка оцінюється на основі попередніх ітерацій [13-15].

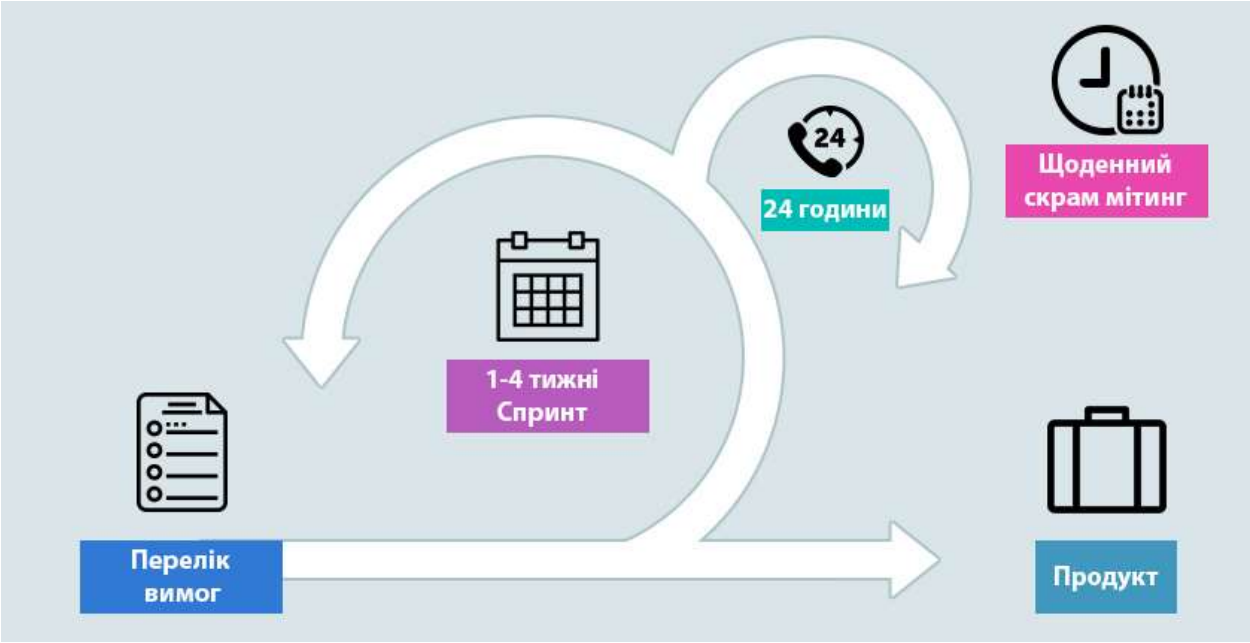

Рис. 1.7. Agile-підхід

У період виконання спринту команда працює автономно та сфокусовано на досягненні визначеної мети. Важливим аспектом Scrum є щоденні короткі наради – daily meetings або daily stand-ups, під час яких учасники команди обмінюються інформацією про виконані завдання, труднощі та плани на день. Це сприяє прозорості, забезпечує постійний контроль над перебігом робіт і дозволяє своєчасно реагувати на можливі

відхилення чи перепони. Після завершення спринту проводиться демонстрація результатів – Sprint Review, де команда презентує функціональний інкремент продукту, отримує зворотний зв’язок від замовника та оцінює відповідність виконаних робіт запланованим цілям. Після цього відбувається ретроспектива (Sprint Retrospective), під час якої команда аналізує свій процес роботи, визначає сильні та слабкі сторони, обговорює шляхи оптимізації та впровадження покращень у наступному циклі. Саме така регулярність перегляду процесу розроблення забезпечує безперервне вдосконалення методів роботи та підвищення ефективності команди [15, 16].

Scrum передбачає розподіл ролей у команді. Ключовими учасниками є власник продукту (Product Owner), який відповідає за формування вимог, пріоритизацію завдань і взаємодію з замовником, Scrum-майстер, що забезпечує дотримання принципів методології, усуває перешкоди та сприяє ефективній командній роботі, а також безпосередньо команда розробників, яка реалізує поставлені завдання та несе відповідальність за інкремент продукту. Важливо, що Scrum запроваджує модель самоорганізації, де рішення щодо шляхів реалізації задач приймає команда, а не керівництво, що сприяє підвищенню відповідальності, мотивації та продуктивності учасників.

Завдяки своїй структурованості, чітким тайм-боксам та регулярному зворотному зв’язку Scrum забезпечує високу адаптивність до змінних вимог, дозволяє швидко тестувати бізнес-гіпотези, знижує ризики провалу проєкту та сприяє отриманню якісного продукту вже на ранніх етапах. Саме тому ця методологія широко використовується у сфері розроблення програмного забезпечення, особливо у швидкозмінному середовищі та умовах високої невизначеності, притаманних стартапам, інноваційним проєктам і продуктам, орієнтованим на кінцевого користувача. Успішність застосування Scrum значною мірою залежить від зрілості команди, наявності дисципліни, прозорості процесів та активної участі замовника, однак при правильній

організації дозволяє істотно підвищити ефективність та результативність розроблення ПЗ [15-17].

1.3.2. Методологія Kanban

Методологія Kanban ґрунтується на принципах безперервного вдосконалення процесів і прозорої організації роботи над проєктом. Основою Kanban є спеціальна канбан-дошка, на якій візуалізуються всі етапи виконання робіт, наприклад «Планування», «Розроблення», «Тестування», «Готово» (рис. 1.8). Завдання відображаються у вигляді карток, які переміщуються між колонками відповідно до поточного етапу їх виконання. Особливістю Kanban є відсутність жорстко визначених ітерацій: нові завдання можуть бути додані в систему у будь-який момент. Робочий процес контролюється шляхом обмеження кількості завдань у роботі на кожному етапі, що дозволяє уникнути перевантаження команди та забезпечити збалансовану продуктивність. Завершення завдання визначається моментом зміни його статусу на «виконано», незалежно від встановлених часових обмежень. Kanban належить до методів Lean-підходу, орієнтованого на усунення надлишкових витрат, підвищення ефективності й швидку реакцію на зміни вимог замовника [17].

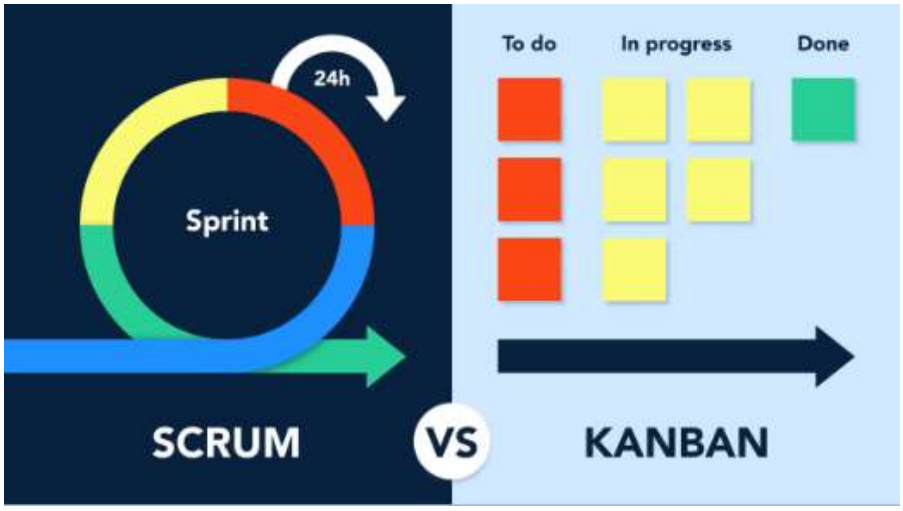

Рис. 1.8. Приклад Kanban дошки

1.3.3. Методологія RUP

RUP (Rational Unified Process) є структурованою ітеративною методологією розроблення програмного забезпечення, яка охоплює повний життєвий цикл проєкту та базується на кращих промислових практиках. Процес складається з чотирьох основних фаз: початкова стадія, уточнення, побудова та впровадження [17-20]. Кожна фаза включає одну або декілька ітерацій, у межах яких виконуються певні завдання – від визначення вимог і ризиків до створення архітектури, розроблення функціонального продукту та його розгортання. Методологія орієнтована на ретельне моделювання та документування, активне управління ризиками й поступове розширення функціональності системи. RUP характеризується масштабністю та універсальністю, тому часто застосовується у великих корпоративних проєктах, де важливі передбачуваність, формальна структурованість і відповідність вимогам бізнесу. Методологія вважається складною для повного охоплення в межах одного короткого опису, оскільки включає широкий набір процесів і рекомендацій, сформованих на основі статистично успішних комерційних проектів [18].

1.3.4. Методологія RAD

RAD (Rapid Application Development) – це методологія швидкого розроблення програмного забезпечення, яка передбачає активне застосування засобів прототипування, інструментальних середовищ і швидких циклів створення продукту [18, 19]. Основна ідея підходу полягає у швидкій розробці діючого прототипу, який дозволяє замовнику оцінити функціонал, надати зворотний зв’язок і коригувати вимоги безпосередньо у процесі створення програмного продукту. У команду зазвичай входить невелика кількість фахівців, а замовник бере активну участь у всіх етапах проєкту. Термін реалізації проєкту за RAD зазвичай не перевищує декількох місяців.

Цей підхід дає змогу значно зменшити часові та фінансові витрати, забезпечуючи високу швидкість реакції на зміни потреб користувачів. Основою RAD є чітке та максимально повне визначення вимог на початковому етапі, при цьому залишається можливість їх коригування у міру розвитку продукту. Методологія ефективна для створення систем із динамічними вимогами та обмеженими термінами розроблення.

1.3.5. Методологія XP

Extreme Programming (XP) – це гнучка методологія, орієнтована на забезпечення високої якості програмного продукту та максимально швидкого реагування на зміну потреб кінцевих користувачів. XP передбачає часті релізи працюючого ПЗ, тісну взаємодію з замовником та активну участь користувача у визначенні функціональних пріоритетів. Ключові технічні практики XP включають парне програмування, безперервну інтеграцію, простоту архітектури, рефакторинг коду та написання тестів до розроблення основної логіки (TDD). Методологія передбачає виконання одного й того ж завдання одночасно декількома розробниками, що сприяє підвищенню надійності та узгодженості коду. Особлива увага приділяється запобіганню помилок на ранніх етапах, що дозволяє швидко створювати якісні програмні продукти. XP найефективніше застосовується в умовах динамічних вимог, активного зворотного зв’язку від клієнта та необхідності швидких результатів [19].

1.4. Порівняння методологій розробки ПЗ

Традиційні моделі, такі як водоспадна передбачають послідовну реалізацію етапів із чітко визначеною документацією, що забезпечує передбачуваність, але обмежує можливість внесення змін. У противагу цьому, гнучкі методології – Agile, Scrum, Kanban, XP – орієнтовані на

швидку адаптацію, ітеративність та постійний зворотний зв’язок із замовником.

Agile виступає концептуальним підходом, у межах якого Scrum, Kanban та XP є конкретними реалізаціями. Основна увага приділяється командній взаємодії, швидкому отриманню результатів, адаптації до змін та мінімізації зайвих процесів. Команда працює невеликими ітераціями, кожна з яких завершується аналізом результатів і коригуванням наступних дій. Такий підхід дозволяє знизити ризик створення продукту, який не відповідає очікуванням користувачів, але при цьому потребує високої професійної зрілості розробників і активної участі замовника.

Scrum, як одна з найпоширеніших Agile-методологій, базується на структурованому циклі коротких спринтів. Перед початком кожного спринту команда проводить планування, формує перелік завдань і встановлює критерії їх завершення. Прозорість процесів, регулярні стендап-зустрічі та ретроспективи сприяють постійному вдосконаленню продукту й командної роботи. Scrum забезпечує баланс структури та гнучкості, що робить його ефективним у командах із середнім та високим рівнем зрілості, особливо в проєктах зі змінними вимогами.

На відміну від Scrum, методологія Kanban не обмежує команду фіксованими інтервалами часу, а організовує роботу як безперервний потік. Головним інструментом є канбан-дошка, на якій відображається стан завдань від планування до завершення. Можливість додавати завдання в будь-який момент і обмеження кількості активних задач дозволяють ефективно балансувати навантаження, усувати «вузькі місця» в процесі та підвищувати продуктивність. Kanban особливо доречний у підтримці систем, DevOps-процесах та середовищах із постійним потоком вимог [18].

RUP – ітеративна методологія корпоративного рівня, що поєднує дисципліну класичних моделей з адаптивністю гнучких методів. RUP містить чіткі фази: початкову, уточнювальну, будівельну та впроваджувальну, причому кожна з них може включати декілька ітерацій. Значна увага

приділяється моделюванню, документації, управлінню ризиками та формалізованим процесам. Це робить RUP ефективним для великих проєктів, але вимагає значних організаційних зусиль та кваліфікації персоналу [19].

RAD зосереджується на швидкому створенні продуктів за допомогою інструментів прототипування та активної комунікації із замовником. Модель орієнтована на короткі цикли, мінімізацію затрат часу та коштів, а також максимальне залучення користувачів. RAD практичний для невеликих команд і проєктів із чітко визначеними цілями, однак може бути недостатнім для великих систем, що вимагають складної архітектури або суворої стандартизації.

XP є найбільш техніко-орієнтованим представником Agile-підходів. Методологія наголошує на високій якості коду, безперервному тестуванні, рефакторингу, створенні простої архітектури та парному програмуванні. Постійна комунікація з користувачем і швидке реагування на зміни забезпечують оперативне виявлення та усунення дефектів. XP надзвичайно ефективна для високодинамічних проєктів, але вимагає високої технічної кваліфікації команди та дисципліни [20].

Узагальнюючи, можна зазначити, що кожна методологія має власні сильні сторони та обмеження. Традиційні підходи забезпечують передбачуваність і структурованість, тоді як гнучкі методи надають адаптивність і швидкість реакції на зміну вимог, тому проєктні команди все частіше вибирають саме гнучкі підходи при розробці ПЗ.

Зведена інформація, щодо гнучких методологій розробки програмного забезпечення наведена в табл. 1.1.

Таблиця 1.1.

Порівняльна таблиця гнучких методологій розробки програмного забезпечення

Критерій / Методологія	**Agile**	**Scrum**	**Kanban**	**RUP**	**RAD**	**XP (Extreme Programming)**
1	2	3	4	5	6	7
Тип методології	Гнучкий підхід	Agile-фреймворк	Lean-Agile метод	Ітераційно-інкрементна модель	Швидка розробка	Agile-метод
Орієнтація	Адаптація, взаємодія, швидка доставка	Ітеративність, самоорганізація команди	Безперервний потік	Дисциплінована розробка, документація	Швидке створення прототипів	Висока якість коду і часті релізи
Тривалість ітерацій	1–4 тижні	1–4 тижні (спринт)	Відсутні, безперервний процес	2–6 тижнів	2–4 місяці	1–2 тижні, іноді щоденні релізи
Робота із завданнями	Фокус на цінності	Product Backlog + Sprint Backlog	Канбан-картки (WIP-ліміти)	Документовані вимоги	Прототипи, моделі	Маленькі задачі, тестування
Гнучкість вимог	Дуже висока	Висока	Дуже висока	Середня	Висока	Дуже висока
Документація	Мінімальна	Мінімальна, спрощена	Мінімальна	Докладна	Середня	Мінімальна
Основна ідея	Швидка адаптація та співпраця	Ітерації (спринти) + ролі	Візуалізація потоку робіт	Стандартизовані процеси	Швидкість + прототипи	Якість + контроль коду
Коли застосовується	Змінні вимоги	Динамічні проєкти	Постійний потік задач	Великі проєкти, корпоративні системи	Невеликі, швидкі проєкти	Інтенсивні проєкти з високою зміною вимог

Продовження таблиці 1.1.

1	2	3	4	5	6	7
Переваги	Швидкість, гнучкість, мінімізація ризиків	Чітка структура, регулярна комунікація	Прозорість, оптимізація процесів	Передбачуваність, стандарти	Дуже швидка розробка	Якість коду, мінімізація дефектів
Недоліки	Потребує зрілої команди	Не працює без дисципліни	Не підходить для хаосу	Висока складність і вартість	Непридатний для великих систем	Високі вимоги до команди
Роль замовника	Активна участь	Постійна взаємодія	Регулярний фідбек	Формалізована участь	Постійна взаємодія	Постійна участь
Результат	Інкремент продукту	Потенційно готовий продукт кожен спринт	Постійний потік готових задач	Повноцінне ПЗ з документацією	Прототип → продукт	Стабільні часті релізи

1.5. Висновки до розділу 1

У першому розділі було проведено комплексний аналіз основних моделей життєвого циклу програмного забезпечення, а також розглянуто еволюцію підходів до організації процесу його розроблення — від класичних каскадних стратегій до сучасних гнучких методологій.

Проаналізовано традиційні моделі, такі як водоспадна, V-подібна, інкрементна та спіральна, що передбачають послідовне або поетапне виконання робіт. Було встановлено, що їхньою перевагою є чіткість структури, можливість контролю на кожному етапі та висока формалізація процесів. Водночас, недоліками таких моделей є низька гнучкість, складність внесення змін у пізніх стадіях розроблення та ризики невідповідності кінцевого продукту актуальним вимогам замовника.

Особливу увагу приділено гнучким методологіям розробки програмного забезпечення (Agile), серед яких детально розглянуто Scrum, Kanban, RUP, RAD та Extreme Programming. Зазначено, що їхня ключова відмінність полягає у здатності швидко адаптуватися до змін, орієнтації на командну співпрацю, ітеративність розроблення та забезпечення безперервного зворотного зв’язку між розробниками та замовником.

У результаті порівняльного аналізу встановлено, що гнучкі методології мають значні переваги у динамічному середовищі сучасних IT-проєктів, де вимоги можуть змінюватися навіть у процесі розробки. Вони дозволяють підвищити ефективність командної роботи, скоротити час виходу продукту на ринок (time-to-market) та підвищити якість кінцевого рішення завдяки постійній взаємодії із замовником і швидкому реагуванню на зворотний зв’язок.

РОЗДІЛ 2
АНАЛІЗ ВПЛИВУ ГНУЧКИХ МЕТОДОЛОГІЙ НА ЕФЕКТИВНІСТЬ КЕРУВАННЯ ПРОЄКТАМИ

2.1. Процеси керування програмними проєктами

Процеси керування програмними проєктами є однією з ключових складових успішної реалізації будь-якої ініціативи зі створення програмного забезпечення. Хоча методологічні підходи можуть суттєво відрізнятися – від класичних моделей до сучасних гнучких методів – сутність роботи менеджера полягає в забезпеченні узгодженої роботи команди, оптимальному використанні ресурсів, контролі строків, вартості та якості кінцевого продукту. Незважаючи на спільні принципи організації, конкретні дії керівника проєкту, обсяг його відповідальності та спосіб взаємодії з командою істотно залежать від внутрішньої культури компанії, специфіки продукту, вимог замовника та використовуваної моделі життєвого циклу. Саме тому неможливо створити універсальний алгоритм управління, який би повністю враховував усі нюанси. Проте існує низка усталених процесів і функцій, які є характерними для більшості програмних проєктів, незалежно від обраної методології [20-21].

Початкова фаза будь-якої розробки передбачає формування взаєморозуміння між замовником і командою розробників щодо мети, очікувань та результатів. У традиційних підходах цей етап часто реалізується у форматі детального формування технічного завдання, узгодження комерційної пропозиції, опису плану виконання та обсягів робіт. Менеджер у цьому процесі відіграє роль комунікатора, аналітика й координатора: він структурує потреби клієнта, перетворює їх на конкретний опис вимог, оцінює масштабність задач, формує попередню оцінку вартості та строків, а також визначає, чи потребує проєкт залучення зовнішніх фахівців або партнерів. У гнучких методологіях також відбувається етап з’ясування потреб, однак

вимоги на ньому фіксуються не у вигляді повного та незмінного документа, а формуються у форматі високорівневого product backlog, що деталізується впродовж проєкту. Це змінює характер управління – замість однократного формування вимог менеджер постійно підтримує їх актуальність, реагуючи на зміни пріоритетів клієнта [22, 23].

Формування команди – ще одна фундаментальна функція керівника. Незалежно від обраного підходу, саме менеджер зазвичай відповідає за підбір учасників, аналіз необхідних компетенцій та розподіл ролей. Однак реальність ринку IT часто обмежує можливість запросити фахівців ідеального рівня: бюджет проєкту, кадрова ситуація на ринку, внутрішня політика організації та потреба розвивати молодих спеціалістів можуть впливати на склад команди. Тому менеджери змушені працювати з доступними ресурсами, компенсуючи нестачу досвіду у частини команди за рахунок наставництва, розподілу функцій або навчання під час проєкту. У традиційних підходах акцент робиться на стабільності складу команди та чіткому визначенні відповідальності кожного учасника. У гнучких методологіях, зокрема Scrum, передбачається самостійність команди у розподілі задач, тоді як менеджер або скрам-майстер створює потрібні умови та усуває перешкоди, що заважають продуктивній роботі. У таких умовах управління людьми набуває більш менторського та фасилітаційного характеру, акцентуючи увагу на мотивації, командній співпраці та професійному розвитку.

Планування проєкту також суттєво трансформується залежно від методології. У традиційних системах цей етап передбачає створення детального графіка робіт, визначення критичного шляху, ресурсного навантаження та плану контролю якості. Менеджер складає календарну модель проєкту, яка є орієнтиром для всіх учасників. Порушення строків у такій системі є критичним, тому управління ризиками та ретельне прогнозування стають невід’ємною частиною плану. У гнучкому середовищі планування має ітеративний характер. Команда оцінює обсяг роботи для

найближчої ітерації, визначає пріоритети, формує sprint backlog і працює над задачами в межах спринту. Загальний план проєкту лишається орієнтовним і може змінюватися під впливом нових вимог, тестування продукту з користувачами або бізнес-пріоритетів. Це дозволяє швидше реагувати на зміни, але одночасно вимагає вищої дисципліни та прозорості в управлінні – без постійного аналізу та актуалізації backlog проєкт може втратити напрям.

Моніторинг і контроль – ще один важливий аспект, який має суттєві відмінності між класичним і гнучким підходом. У традиційних моделях відстеження прогресу зазвичай ґрунтується на формальних звітах, періодичних нарадах та порівнянні фактичних результатів із запланованими показниками. У деяких випадках це може призводити до затримки виявлення проблем, адже звітність часто створюється з певним часовим лагом. Проте системний моніторинг допомагає утримати проєкт у межах контрольних точок і забезпечує передбачуваність результату. У гнучких методологіях контроль є безперервним і відбувається через щоденні зустрічі, огляд результатів кожної ітерації та використання візуальних інструментів – діаграм згоряння задач, Kanban-дощок, графів швидкості виконання. Постійне спілкування з командою дозволяє виявляти проблеми на ранніх етапах та швидко їх вирішувати, а прозорість процесів підвищує відповідальність кожного учасника. У такій системі менеджер менше покладається на формальні документи й більше на живу комунікацію, спостереження та регулярну оцінку робочого темпу [23].

Важливо також розуміти, що життєвий цикл великих програмних проєктів може розтягуватися на роки. За цей час іноді змінюються вимоги, бізнес-цілі або ринкове середовище, через що виникає потреба адаптувати початкову концепцію продукту. У деяких випадках зміни настільки суттєві, що модернізувати систему стає дорожче і складніше, ніж розпочати її розробку з нуля. Тому правильне управління вимогами і своєчасна адаптація планів мають вирішальне значення для успішної реалізації проекту [24, 25].

Ще однією обов’язковою функцією є комунікація із зацікавленими сторонами. Менеджер формує регулярні звіти, описує стан проєкту, ризики, виконані етапи та плани на майбутнє. Звітність має бути лаконічною і зрозумілою, містити ключові індикатори виконання та забезпечувати прозорість процесу для керівництва компанії, замовника або партнерів. Такі звіти стають основою для прийняття рішень і слугують інструментом контролю над динамікою проекту [25-27].

Важливим концептуальним елементом управління є так званий «проєктний трикутник», який поєднує три основні параметри – час, бюджет і обсяг робіт. Між цими характеристиками існує сильна взаємозалежність: зміна одного з них практично завжди впливає на два інші. Наприклад, скорочення строків виконання може вимагати збільшення фінансування або зменшення обсягу задач. Водночас зменшення бюджету змушує відмовлятися від певних функцій або розширювати часові рамки. Ця модель відображає практичну необхідність постійного балансування між інтересами замовника, можливостями команди та обмеженнями ринку. У центрі трикутника традиційно розташовують показник якості – він залежить від усіх інших компонентів і є вразливим до будь-яких коливань у плані [1, 22, 28].

Рис. 2.1. Проектний трикутник

Керування цим балансом вимагає ретельного планування та здатності передбачати наслідки кожного рішення. Наприклад, якщо розробники викликають випередження графіка, керівник може інвестувати додатковий час у тестування, підвищуючи якість продукту. Натомість скорочення бюджету часто змушує оптимізувати функціонал або пришвидшувати роботу, що потенційно погіршує якість кінцевого результату. Тому головне завдання менеджера полягає у побудові структури робіт, яка одночасно забезпечить дотримання строків, раціональне використання бюджету та створення продукту, що відповідає очікуваному рівню якості [28, 29].

Для забезпечення такого балансу традиційне управління використовує детальні графіки, розбиття робіт на етапи та контрольні точки. Проєктний план у такому середовищі виступає фундаментом всієї діяльності: він має включати завдання на кожному етапі життєвого циклу, визначати очікувані результати, ресурси, відповідальних осіб і критерії завершення. Чітка структурованість дозволяє не лише прогнозувати майбутні потреби, а й завчасно визначати потенційні ризики, готувати превентивні дії та мінімізувати вплив непередбачених труднощів [29, 30].

Особливе місце у керуванні проєктом займає робота з ризиками. Незалежно від підходу, менеджеру необхідно прогнозувати потенційні проблеми та мінімізувати їх вплив на проєкт. У класичних моделях ризики визначаються на старті, оцінюються за рівнем ймовірності та впливу, і на основі цього створюється план реагування. У гнучких підходах ризик-менеджмент не є одноразовою процедурою – це безперервний процес, що враховується під час кожної ітерації. Часті релізи, інкрементне тестування та короткі цикли зворотного зв’язку дозволяють виявляти невизначеності швидше. Таким чином гнучкі методи знижують імовірність виникнення критичних прорахунків у пізніх стадіях проєкту, але вимагають значно активнішої участі менеджера та команди у прогнозуванні змін.

Оцінювання вартості та ресурсів – ще одна важлива функція управління. У традиційних системах бюджет зазвичай фіксується на початку

проєкту, а його перевищення є небажаним і часто неприпустимим. Гнучке середовище оперує змінним обсягом роботи при фіксованому бюджеті або фіксованому часі. У такій ситуації замовник отримує можливість впливати на функціональність продукту, жертвуючи частиною вимог заради дотримання строків і бюджету. Це робить фінансову модель гнучких методів більш адаптивною, але вимагає постійної комунікації з бізнес-сторони й обґрунтування рішень щодо пріоритетів. Менеджер виступає не тільки контролером витрат, а й стратегом, який разом із замовником формує найцінніший набір функцій у межах доступних ресурсів [30].

Документування й звітність також є невіддільною частиною управління. У традиційних моделях створюються великі формальні документи – контракти, специфікації, статус-звітність, аналітичні огляди, акти виконаних робіт. У гнучких методологіях документація залишається, але її форма відрізняється: замість великих формалізованих звітів використовуються короткі інформативні огляди, артефакти спринтів, backlog-документи, діаграми прогресу. Основний акцент переноситься на прозорість і доступність інформації, а не на формальність її подачі. Таким чином менеджер у гнучкому середовищі приділяє більше уваги комунікації та координації роботи, ніж документальному контролю.

Окремо слід зазначити роль змін у проєкті. У традиційних системах зміни часто сприймаються як небажані, оскільки вони порушують заздалегідь складений план, збільшують вартість і можуть впливати на строки. У гнучких підходах зміни є природним явищем: передбачається, що продукт еволюціонує відповідно до потреб бізнесу, тому процес управління має забезпечувати здатність швидко адаптуватися. Менеджер у такій системі стає каталізатором гнучкості, допомагаючи команді адаптувати плани та перебудовувати робочий процес у відповідь на нові вимоги.

Усі наведені аспекти демонструють, що процеси управління в гнучких і класичних моделях мають різну природу. Традиційні підходи спираються на передбачуваність, плановість і контроль, тоді як гнучкі – на адаптивність,

швидкість зворотного зв’язку та тісну взаємодію з командою. Проте обидві парадигми мають спільну мету – забезпечити успішне завершення проєкту, тому сучасна практика часто використовує їх комбінування залежно від масштабів і контексту. Гнучкі методи значно підвищують можливість швидко реагувати на зміни, зменшують ризики великих помилок на пізніх етапах і дають змогу створювати продукт, максимально наближений до потреб користувача. Однак для ефективності вони потребують високої зрілості команди, культури відкритої комунікації та значної залученості замовника. Класичні підходи натомість забезпечують передбачуваність і структурованість, що є важливим для великих програмних продуктів, державних проєктів та сфер із фіксованими вимогами [29-31].

Таким чином, вплив гнучких методологій на управління проєктами полягає передусім у зміні ролі менеджера й перефокусуванні організаційної логіки: від централізованого контролю – до підтримки, координації, мотивації та стратегічного спрямування. Ефективність керування в таких умовах зростає за рахунок прозорості процесів, регулярного аналізу стану продукту, швидкої реакції на відхилення та підвищеної відповідальності команди за результат. Гнучкі підходи дозволяють уникати накопичення проблем, підсилюють адаптивність організації та сприяють створенню конкурентних цифрових продуктів у динамічному ринковому середовищі.

2.2. Процес планування в умовах використання гнучких методологій

Планування у сфері створення програмних продуктів є фундаментальною складовою управління проєктом та визначає рамки, у яких розгортається подальший життєвий цикл розробки. Незалежно від того, чи використовується традиційна модель, чи застосовуються підходи Agile, процес планування охоплює встановлення часових, бюджетних, кадрових та технічних обмежень, які впливають на подальший розподіл завдань, вибір інструментів і підхід до розробки. Гнучкі методології не відмовляються від

планування, а здійснюють його ітеративно, акцентуючи увагу на адаптації та постійному перегляді планів у відповідь на реальні зміни вимог та середовища [30].

Початковим кроком планування завжди є визначення ключових параметрів проєкту: оцінка обсягу робіт, визначення ресурсів, доступних виконавців, встановлення часових рамок та бюджетних меж. Одночасно формується уявлення про структуру майбутнього продукту та визначаються основні функціональні блоки, що впливають на вибір архітектурних та технологічних рішень. Саме на цьому етапі керівник здійснює розподіл ролей і відповідальностей між учасниками, враховуючи рівень їх кваліфікації, досвід та можливість залучення у визначені періоди [30].

Після формування початкових рамок розробляються орієнтовні етапи виконання робіт та результати, які мають бути отримані після кожного з них. У традиційних моделей це може бути детальний послідовний план, тоді як у гнучких середовищах – набір спринтів або інкрементів, кожен з яких завершується конкретним результатом: прототипом функціоналу, готовою частиною системи або тестованою версією продукту. У будь-якому випадку створюється базова дорожня карта – проектний план, що окреслює кінцеву мету та ключові контрольні точки [31].

Ключовою особливістю планування в розробці ПЗ є його циклічність. Після складання початкового графіка починається періодичний аналіз і звірка між фактичним станом виконання проєкту та початковими прогнозами. У традиційних методах ці ревізії здебільшого проводяться у визначені контрольні моменти, тоді як у гнучких методологіях – у значно частішому ритмі, зазвичай раз на 1–3 тижні в межах спринтів. Під час таких ревізій відстежуються відхилення, порівнюються результати з очікуваннями, проводиться переналаштування пріоритетів і за необхідності вносяться зміни до плану [32].

На практиці трапляється ситуація, коли початкові оцінки виявляються недостатньо точними. Це природно, оскільки програмна інженерія часто має

справу з невизначеними вимогами, еволюційними продуктами та технологічними ризиками. У результаті після отримання перших реальних даних про швидкість розробки, складність завдань та взаємозалежність компонентів, план починає уточнюватися. Якщо зміни впливають на загальні строки або бюджет, обов'язково проводяться перемовини із замовником або стейкхолдерами та здійснюється узгодження нових меж проєкту. У традиційних підходах це є винятковою подією, тоді як Agile вважає подібне уточнення природним робочим процесом.

Світовий досвід показує, що навіть найретельніше складені плани не можуть передбачити всі можливі ризики та непередбачувані ситуації. Зміни в технологічних трендах, виявлення потреби в додаткових компетенціях, коригування вимог замовника або зміни ринку – усе це чинники, що впливають на динаміку проєкту. Тому гнучкість і здатність адаптувати план є критичними, а інколи й визначальними факторами успішності. Саме тому у сучасних підходах планування не розглядається як статичний документ, а як живий процес, що супроводжує весь життєвий цикл проєкту. На рис. 2.2 представлено схематичне відображення процесу планування як багатошарового ітераційного процесу [33].

Зміст проектного плану може відрізнятися залежно від масштабу, типу програмного продукту, корпоративних стандартів та моделі ЖЦ, однак у більшості випадків він містить низку ключових компонентів. Зокрема, вступна частина описує основні параметри проєкту: мету, обґрунтування, часові рамки, доступні ресурси та ключові обмеження. Далі йде розділ оцінки ризиків, у якому аналізуються можливі загрози, що можуть вплинути на виконання проєкту, а також визначаються стратегії їх мінімізації.

Важливою складовою плану є управління людськими ресурсами. Тут визначаються компетентності, необхідні для реалізації проєкту, механізми розподілу ролей, підходи до прийняття рішень та взаємодія між учасниками. Гнучкі методології особливу увагу приділяють самокерованості команд, участі спеціалістів у плануванні та підвищенню їх відповідальності за

результат. Традиційні підходи, навпаки, частіше опираються на ієрархічний розподіл ролей та централізоване прийняття рішень [34].

Рис. 2.2. Етапи процесу планування програмного проєкту

До проектного плану також включаються розрахунки необхідних програмних і апаратних ресурсів, опис інструментів, ліцензій та технічних засобів. Для великих систем розробляється графік закупівель та оновлення обладнання, що впливає на бюджет і строк реалізації проєкту.

Окреме місце займає система контролю прогресу. У плані мають бути визначені критерії досягнення цілей, порядок підготовки звітів, частота звітності та механізми внутрішнього і зовнішнього моніторингу. Навіть у гнучких методологіях, де створюються умови для децентралізованого прийняття рішень, прозорість виконання і наявність контрольних точок є невід’ємною частиною керування. Саме через регулярні рев’ю, демонстрації проміжних результатів та оновлення планів забезпечується керованість процесу.

Невід'ємним інструментом планування є контрольні позначки – моменти, що фіксують завершення певного етапу роботи. Вони дозволяють оцінити стан проєкту, скоригувати темп виконання задач і оновити прогноз завершення. У гнучких підходах контрольні точки часто збігаються із завершенням спринтів або інкрементів, а результати їх досягнення виражаються у формі працюючого функціоналу. У традиційних моделях контрольні позначки зазвичай пов’язані із завершенням фаз, наприклад, аналізу вимог, проєктування архітектури, розробки або тестування [33-35].

Фіксація цих точок (рис. 2.3) є критично важливою не лише для контролю, але й для взаємодії із замовником. Надання проміжних результатів дозволяє узгоджувати очікування, отримувати зворотний зв’язок і здійснювати корекцію продукту, що знижує ризик накопичення невідповідностей і збільшує ймовірність успішного завершення проєкту.

З точки зору управління документацією планування потребує організованого документообігу. Будь-які зміни у графіку, бюджеті, складі ресурсів або обсязі функцій повинні бути зафіксовані, погоджені та доведені до команди. Гнучкі методології мінімізують формальні документи, але не відмовляються від них повністю – натомість вони акцентують увагу на

доступності інформації і швидкому оновленні артефактів планування. У традиційних підходах документація є більш структурованою і детальною, що підвищує передбачуваність, але знижує адаптивність.

Рис. 2.3. Послідовність розробки специфікацій вимог

Таким чином, планування в розробці програмного забезпечення є ітераційним, багаторівневим і динамічним процесом, що забезпечує баланс між прогнозованістю та гнучкістю. Висока якість планування дозволяє попередити ризики, забезпечити оптимальне використання ресурсів і створити умови для успішної реалізації проєкту. А гнучкі методології, завдяки своїй адаптивній природі, перетворюють план на інструмент постійного управління змінами, що дозволяє реагувати на виклики сучасного бізнес-середовища і підвищувати ефективність розробки.

2.3. Планування, побудова та оптимізація календарного графіка робіт проєкту

Планування графіка виконання робіт є однією з фундаментальних складових управління програмним проєктом. Від правильного визначення послідовності робіт, їх тривалості та необхідних ресурсів залежить не лише своєчасність завершення розробки, але й можливість контролю процесу, ефективного розподілу навантаження між учасниками команди та раннього

виявлення ризиків. Менеджер проєкту на цьому етапі фактично формує детальну «дорожню карту» реалізації, що слугує основою для подальшого моніторингу стану робіт [34].

У випадках, коли компанія вже має досвід реалізації подібних рішень, попередні графіки можуть використовуватися як шаблон. Однак навіть при схожості вимог кожен програмний продукт є унікальним – змінюються характеристики замовлення, склад команди, технологічний стек, а також зовнішні фактори. Тому початковий графік завжди потребує адаптації до реальних умов нового проєкту.

Особливої уваги вимагають інноваційні розробки, що не мають аналогів або лише частково базуються на попередньому досвіді організації. У таких випадках стартові оцінки часу та ресурсів, навіть за участі кваліфікованих керівників, можуть бути оптимістичними. Це пояснюється тим, що сучасні програмні проєкти часто стикаються з динамічними змінами вимог, невизначеністю технічних рішень або появою нових факторів у процесі реалізації. Тому актуалізація календарного плану на основі оперативної інформації – обов'язковий елемент адаптивного планування [35].

На рис. 2.4 представлено узагальнену модель процесу формування графіка робіт. Основне завдання – визначення етапів життєвого циклу розробки, приблизної тривалості кожного з них, а також ключових контрольних подій (milestones). Обрана модель життєвого циклу (каскадна, інкрементальна, Agile тощо) суттєво впливає на структуру графіка. Наприклад, в адаптивних підходах значна частина робіт може виконуватися паралельно, що передбачає ефективний перерозподіл ресурсів.

Тривалість окремих етапів визначається з урахуванням складності завдань, потенційних ризиків, обсягу вимог та доступності компетентних фахівців. Мінімальна розумна тривалість задачі зазвичай становить кілька днів, а для великих проєктів – до одного місяця. Занадто короткі інтервали призводять до надто частого оновлення графіка і збільшення

адміністративних витрат, тоді як надмірно довгі – ускладнюють контроль і своєчасне реагування на відхилення.

Рис. 2.4. Процес побудови календарного графіка робіт

У практиці оцінювання тривалості застосовується емпіричне правило: сума ідеальних тривалостей задач збільшується на 30% для врахування можливих проблем, і ще на 20% – як резерв на непередбачувані ситуації. Такий підхід узгоджується з «проєктним трикутником» (див. рис. 2.1), який демонструє залежність між часом, ресурсами та вартістю – збільшення одного параметра зазвичай веде до зростання інших [36].

Для графічного представлення плану використовують кілька видів діаграм: часові графіки, мережеві схеми та діаграми навантаження. Значну частину таких матеріалів можливо синхронізувати з базами даних та формувати автоматично через системи управління проєктами.

Приклад мережевого графіка подано на рис. 2.5. Діаграма демонструє залежності між окремими задачами та дає змогу визначити, які етапи можуть виконуватися паралельно, а які – лише послідовно. Контрольні точки на схемі позначають ключові переходи між фазами проєкту.

Особливе значення має визначення критичного шляху – найдовшого ланцюга пов'язаних завдань, які формують мінімально можливий термін реалізації проєкту. Будь-яка затримка на критичному шляху безпосередньо впливає на загальний строк завершення проєкту. Аналіз критичного шляху дозволяє оптимізувати розподіл ресурсів і зменшити ризик затримок.

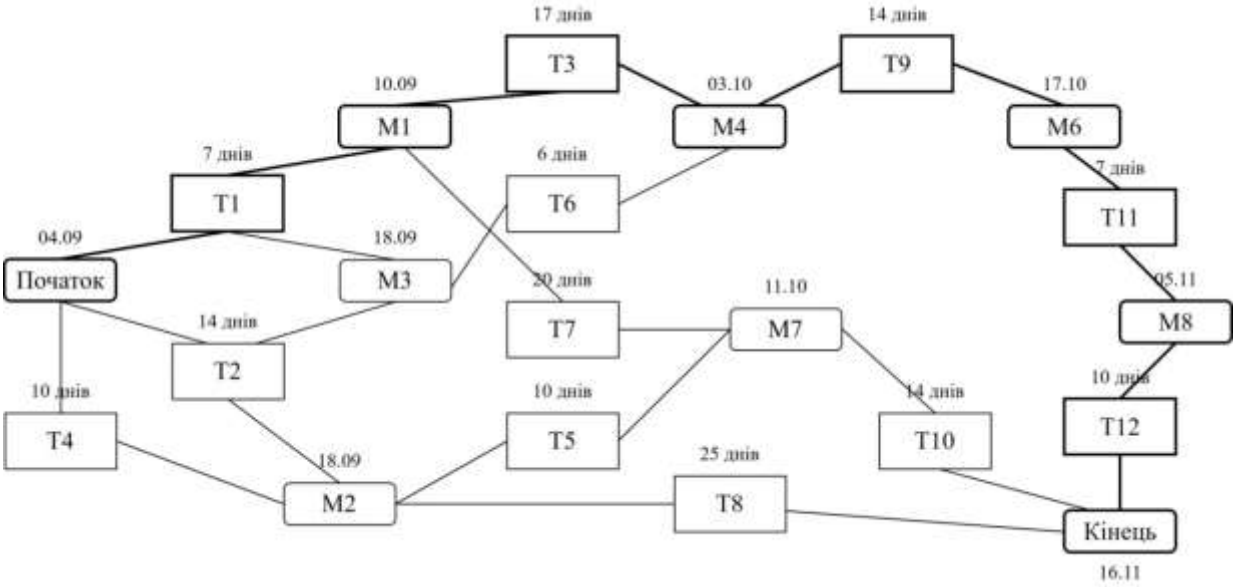

Рис. 2.5. Мережева діаграма виконання робіт

Графічним відображенням календарного плану є діаграма Ґанта, представлена на рис. 2.6. Вона показує часові межі кожного етапу, очікувані запаси часу та резерви (які позначають можливу затримку без впливу на дедлайн проєкту). Елементи без резерву відповідають критичному шляху.

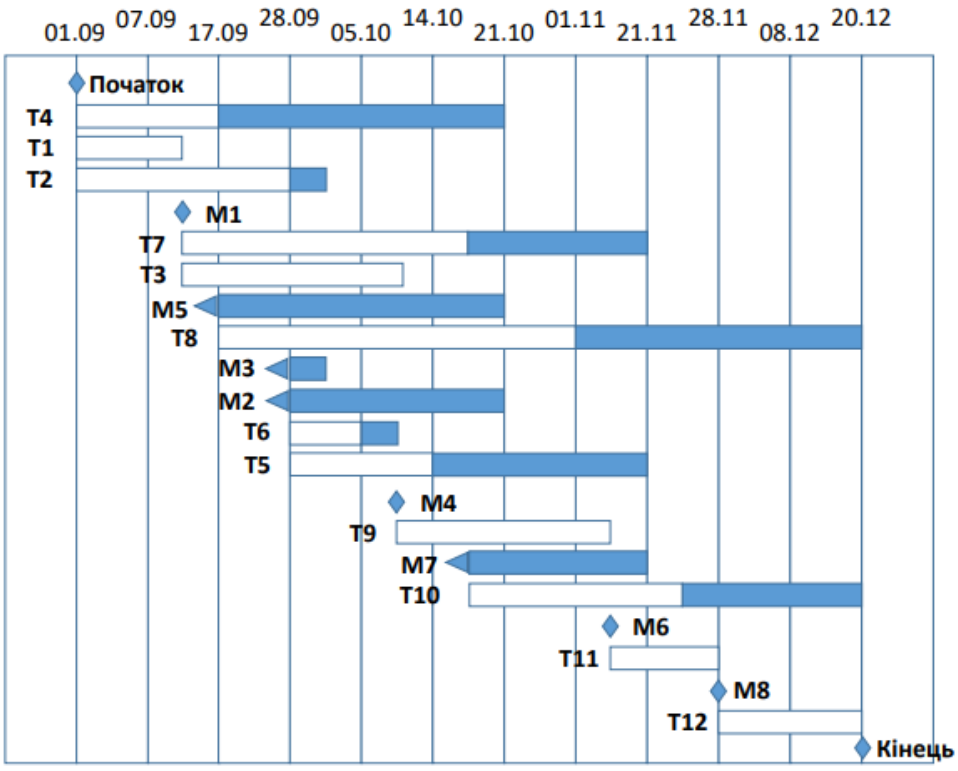

Рис. 2.6. Діаграма Ґанта для проєкту

Окрему важливу роль відіграє діаграма зайнятості ресурсів (рис. 2.7), що дає змогу оцінити ефективність використання фахівців. У складних проєктах члени команди можуть працювати над кількома задачами або декількома проєктами одночасно. Нерівномірне навантаження може призвести як до простоїв, так і до перевантаження виконавців. У таблиці 2.1 наведено приклад призначення виконавців на етапи проекту.

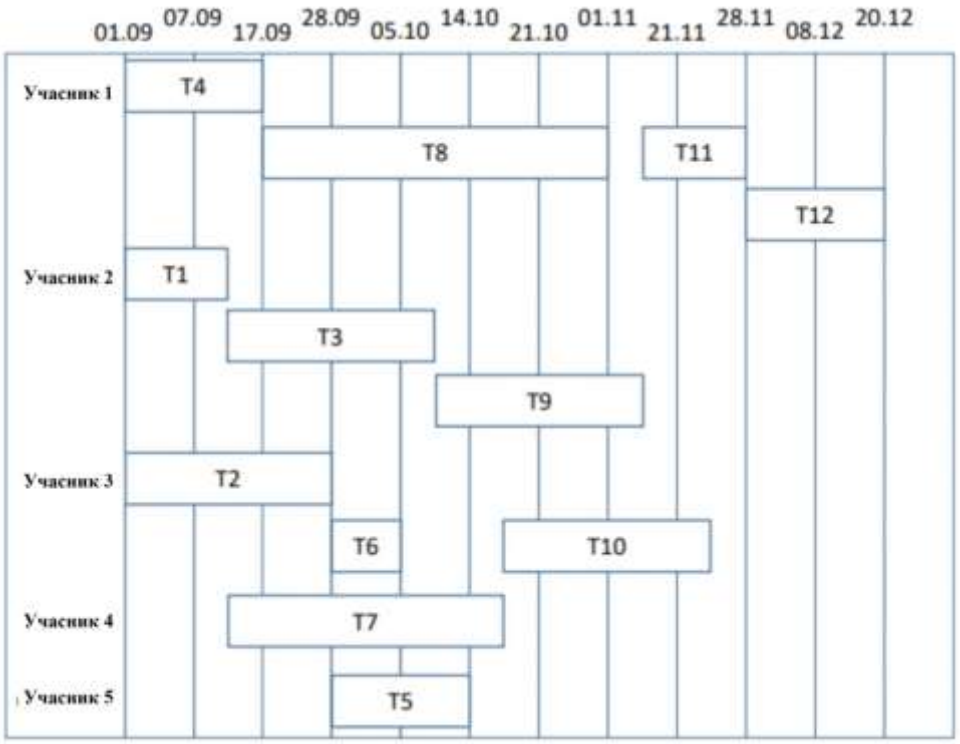

Рис. 2.7. Діаграма зайнятості команди розробників

Початковий графік робіт, як правило, є прогнозним. У процесі реалізації він неодноразово коригується відповідно до фактичної інформації, результатів моніторингу прогресу, оновлення вимог, зміни пріоритетів і доступності ресурсів. Менеджер систематично аналізує відхилення між плановими й реальними строками, вносить зміни і переглядає наступні інтервали роботи, намагаючись скоротити критичний шлях і забезпечити завершення проєкту в заплановані строки.

Таблиця 2.1.

Приклад призначення виконавців на етапи проєкту

Етап	Тривалість етапу	Виконавець
T1	10	Учасник 2
T2	28	Учасник 3
T3	30	Учасник 2
T4	17	Учасник 1
T5	20	Учасник 5
T6	7	Учасник 3
T7	35	Учасник 4
T8	44	Учасник 1
T9	20	Учасник 2
T10	10	Учасник 3
T11	10	Учасник 1
T12	23	Учасник 1

Розглянемо іще приклад практичного застосування інструментів управління проектами, зокрема мережевого графіка та діаграми Ґанта, для відображення процесу планування розроблення програмного забезпечення.

У даній ситуації компанія, що працює у штатному режимі, отримала нове замовлення на розроблення програмного продукту. Для ефективного планування та контролю ходу робіт необхідно побудувати мережеву модель проєкту та відповідну діаграму Ґанта, які дозволяють оцінити послідовність виконання завдань, визначити критичні шляхи та забезпечити оптимальний розподіл ресурсів.

Початковий мережевий графік подано у вигляді орієнтованого графа з двома ключовими вершинами, які позначають початок та завершення робіт (рис. 2.8). У вершинах графа наведено події, що відповідають основним етапам виконання проекту, а стрілками позначено види робіт, які з’єднують відповідні події [36].

У межах цього прикладу розглядаються такі події:

- початок виконання робіт;
- підготовка робочих місць і формування команди виконавців;
- завершення етапу високорівневого проєктування програмного забезпечення;

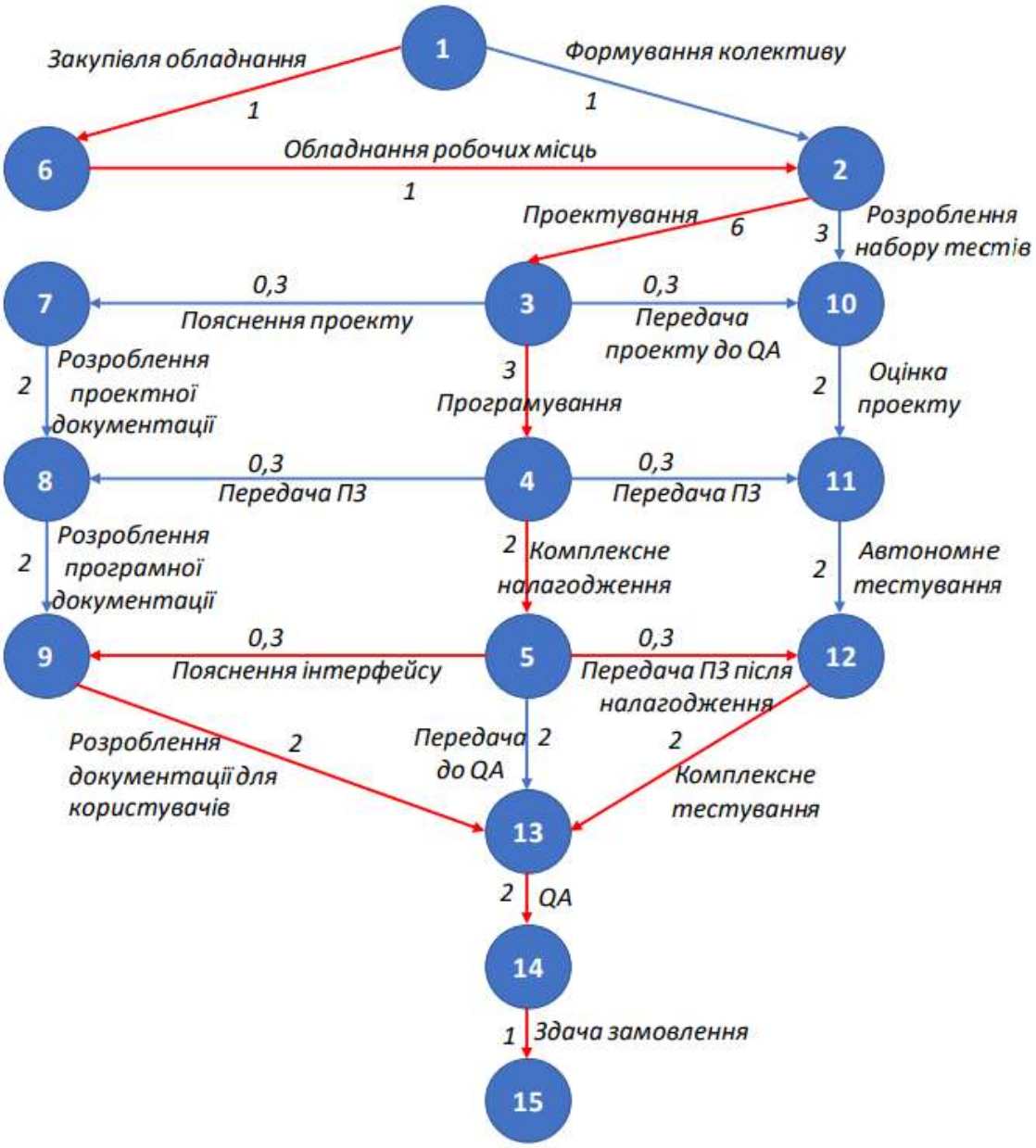

Рис. 2.8. Мережевий графік процесу планування розроблення програмного забезпечення

- завершення детального проєктування та програмування;
- завершення комплексного налагодження програмного продукту;
- закупівля необхідного обладнання;
- отримання групою з документування опису проєкту та пояснень від проектувальників;
- завершення створення проектної документації;

- підготовка користувацької документації та завершення її розроблення;
- розроблення тестів групою оцінки якості;
- позитивна оцінка проєкту групою контролю якості;
- завершення автономного тестування;
- завершення комплексного тестування та підготовка фінальної документації;
- завершення перевірки якості;
- завершення робіт над проектом.

Під кожною стрілкою на рис. 2.8 наведено тривалість відповідної роботи у тижнях. Аналіз побудованого графіка дозволяє визначити два критичних шляхи: 1–6–2–3–4–5–12–13–14–15 та 1–6–2–3–4–5–9–13–14–15.

Це свідчить про те, що завершити проєкт раніше ніж за 18,3 тижня неможливо, оскільки будь-яка затримка на критичному шляху призведе до зсуву кінцевих термінів виконання.

З точки зору оптимізації процесу, можна спробувати перебудувати мережеву діаграму так, щоб усі можливі шляхи між початковою та кінцевою подіями мали приблизно однакову тривалість. Наприклад, доцільно залучити групу оцінки якості до тестування вже на ранніх етапах, що зменшить навантаження на розробників. Водночас це може призвести до зниження якості кінцевого продукту, адже прискорення процесу часто впливає на глибину перевірки (згідно з принципом «проектного трикутника» – рис. 2.1). У разі виявлення помилок після тестування час на їх виправлення збільшує загальну тривалість проекту.

На практиці, особливо у великих проектах з численними взаємопов’язаними етапами, удосконалення мережевого графіка досягається завдяки більш ефективному розподілу завдань і ресурсів між командами та етапами робіт [37].

Більш детальний аналіз мережевого графіка показує, що події 1, 2 та 6 заплановано не цілком раціонально:

- команда розробників формується за один тиждень, однак робочі місця на цей момент ще не підготовлені;
- спеціалісти з документування починають роботу лише через шість тижнів після проектувальників;
- група оцінки якості має майже тиждень простою до завершення проектування.

Такі проблеми є типовими для реальних IT-проектів і вирішуються кожною компанією індивідуально – шляхом гнучкого планування, перерозподілу ресурсів або зміни черговості робіт.

Для порівняння побудовано діаграму Ґанта (рис. 2.9), яка дає змогу наочно відобразити часові межі виконання кожного завдання, їх послідовність і взаємозв’язки. Діаграма Ґанта також дозволяє визначити критичні точки проєкту, періоди простоїв та дублювання завдань, що робить її ефективним інструментом для моніторингу ходу розроблення програмного забезпечення.

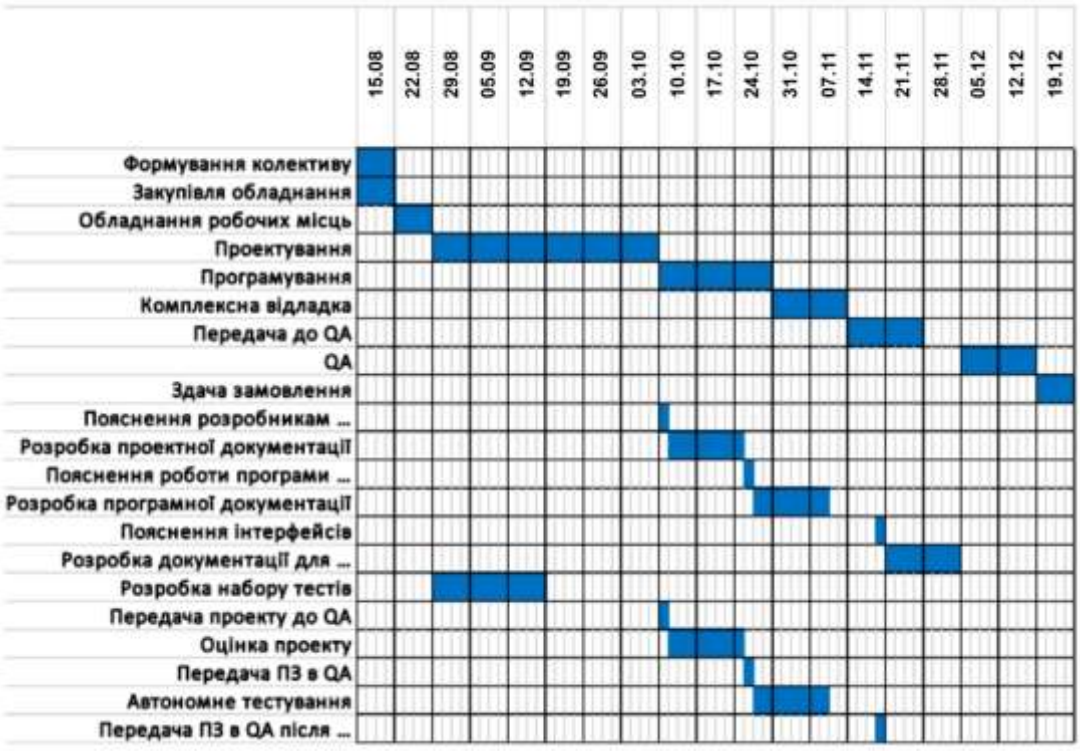

Рис. 2.9. Діаграма Ґанта для проєкту розроблення програмного забезпечення

Мережева діаграма дозволяє наочно простежити взаємозв'язок між окремими етапами виконання проєкту, тоді як діаграма Ґанта відображає динаміку робіт у часі, тобто показує, які саме дії здійснюються у кожен конкретний момент. Наприклад, з аналізу діаграми Ґанта видно, що між етапами 11–12 та 12–13 група оцінки якості має перерву в роботі тривалістю майже два тижні.

Саме завдяки такій візуальній наочності керівники проєктів часто надають перевагу саме діаграмі Ґанта, оскільки вона дає змогу чітко відстежувати послідовність дій, визначати завантаження персоналу та планувати використання ресурсів у часі. Крім того, над кожним відрізком роботи на цій діаграмі можна вказати кількість виконавців, задіяних у виконанні певного завдання, що робить інструмент особливо зручним для управління людськими ресурсами.

На відміну від цього, мережева діаграма є менш зручною для відображення кількісних параметрів – таких як тривалість роботи або чисельність виконавців, – проте вона має інші переваги. Технічні менеджери віддають перевагу саме мережевим графікам, оскільки вони дають змогу чітко відстежити взаємозалежність робіт і визначити критичні шляхи проєкту. Ці дані є надзвичайно важливими для оперативного контролю та перегляду планів, адже критичні шляхи часто потребують регулярного оновлення у процесі реалізації проекту [36].

Отже, підсумовуючи розгляд основ управління проєктом, можна зробити такі узагальнення:

1) ефективне управління програмним проєктом характеризується виконанням робіт відповідно до встановленого графіка та в межах запланованого бюджету;

2) управління програмними проєктами має свої особливості порівняно з іншими видами технічних проєктів, оскільки програмне забезпечення є нематеріальним продуктом. Досвід, отриманий під час реалізації попередніх

проєктів, не завжди можна безпосередньо використати при створенні інноваційних або складних ПЗ-систем;

3) менеджери програмних проєктів виконують широкий спектр завдань, серед яких ключовими є: планування, оцінювання необхідних ресурсів і формування графіка робіт. Ці процеси мають ітераційний характер, тобто тривають протягом усього життєвого циклу проєкту. У міру надходження нової інформації щодо його виконання плани та графіки підлягають перегляду і коригуванню;

4) контрольні точки (або контрольні позначки) – це прогнозовані результати завершення певних етапів реалізації проєкту. Разом із короткими звітами вони передаються керівництву для моніторингу ходу виконання робіт. У свою чергу, контрольні проєктні елементи є глобальними контрольними точками, які призначені для звітування перед замовником програмного продукту;

5) побудова графіка робіт включає створення різних графічних представлень плану проєкту, серед яких особливе місце займають мережеві діаграми – для відображення взаємозалежності етапів, – та часові (стрічкові) діаграми, що демонструють тривалість виконання кожного з них.

2.4. Висновки до розділу 2

У другому розділі було здійснено поглиблений аналіз впливу гнучких методологій розроблення програмного забезпечення на ефективність процесів управління проєктами. Розглянуто основні принципи організації управління в межах Agile-підходів, а також проведено порівняння традиційних ітераційних моделей із сучасними методами планування, контролю та координації робіт у команді.

Показано, що використання гнучких підходів сприяє підвищенню ефективності проєктного менеджменту завдяки таким чинникам, як прозорість процесів, постійний моніторинг виконання завдань, залучення

всіх учасників команди до прийняття рішень та безперервна адаптація планів до змін зовнішнього середовища. Зокрема, методології Scrum і Kanban забезпечують гнучке планування спринтів, візуалізацію робочого процесу за допомогою дошок завдань, що значно полегшує управління пріоритетами й ресурсами.

Особливу увагу приділено процесу планування в умовах гнучких методологій, який відрізняється динамічністю, короткими циклами зворотного зв’язку та можливістю оперативного коригування графіка виконання робіт. Визначено, що в Agile-підходах планування відбувається не лише на початку проєкту, а є безперервним процесом, що супроводжує розроблення продукту на всіх етапах його життєвого циклу.

У роботі також розглянуто методи побудови та оптимізації календарного графіка проєкту, що є ключовими елементами ефективного управління. На основі застосування інструментів візуалізації, зокрема діаграм Ганта, здійснено моделювання процесу планування та контролю завдань. Побудова діаграми Ганта дозволила наочно представити взаємозв’язки між етапами, терміни їх виконання, залежності між задачами, а також визначити критичний шлях проєкту. Це дало змогу оцінити вплив окремих процесів на загальну тривалість виконання робіт та оптимізувати розподіл ресурсів у межах проєкту.

Проведений аналіз довів, що впровадження гнучких методологій в управління програмними проєктами дає змогу не лише підвищити ефективність командної взаємодії, а й забезпечити глибшу прозорість процесів, покращити прогнозування строків виконання завдань та зменшити ризики зриву термінів.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ УПРАВЛІННЯ ПРОЄКТАМИ

3.1. Архітектура системи

Розроблена система управління проєктами реалізована на основі клієнт-серверної архітектури, яка забезпечує високу продуктивність, масштабованість і зручність розширення функціоналу. Такий підхід дозволяє ефективно розподіляти навантаження між клієнтською та серверною частинами, забезпечуючи стабільну роботу навіть за великої кількості одночасних користувачів.

Клієнтський застосунок розроблений з використанням React.js – сучасної JavaScript-бібліотеки для побудови інтерактивних користувацьких інтерфейсів. Для стилізації інтерфейсу застосовується Bootstrap 5 у поєднанні з Tailwind CSS, що забезпечує адаптивність, швидке створення компонентів і єдиний візуальний стиль.

Для керування станом застосунку використовується бібліотека Redux Toolkit, а для асинхронної взаємодії з сервером – Axios, яка забезпечує зручне виконання HTTP-запитів та обробку помилок.

Клієнтський застосунок реалізує основні функції системи:

- авторизацію та реєстрацію користувачів;
- створення та редагування проєктів і завдань;
- керування командами;
- побудову діаграми Ґанта (за допомогою бібліотеки AnyChart);
- відображення статистики про виконання проєктів.

Серверна логіка системи реалізована з використанням Node.js у поєднанні з фреймворком Express.js, який забезпечує просте керування маршрутами, обробку HTTP-запитів і реалізацію REST API.

Для забезпечення безпечної автентифікації користувачів використовується технологія JSON Web Token (JWT), яка дозволяє зберігати

інформацію про права доступу користувача у зашифрованому токені. Це дає змогу безпечно виконувати вхід у систему без постійного збереження облікових даних.

Серверна частина реалізує основну бізнес-логіку, зокрема:

- перевірку автентичності користувача;
- управління базою даних (створення, оновлення, видалення записів);
- обробку запитів клієнта та формування відповідей у форматі JSON;
- журналювання подій і моніторинг стану системи.

Для зберігання інформації про користувачів, команди, завдання, коментарі та проєкти використовується MySQL – потужна реляційна система управління базами даних, яка забезпечує надійність, транзакційність і підтримку складних запитів.

Взаємодія із СУБД здійснюється за допомогою ORM-бібліотеки TypeORM, яка дає змогу працювати з базою даних на рівні об’єктів, спрощуючи розробку та підтримку коду.

Архітектурна взаємодія компонентів:

1) користувач взаємодіє з інтерфейсом, розробленим на React;js;
2) клієнтська частина надсилає HTTP-запит через Axios до REST API серверної частини Express.js;
3) серверна частина обробляє запит, звертається до бази даних MySQL;
4) отримані дані передаються назад клієнту у форматі JSON;
5) React оновлює інтерфейс користувача без перезавантаження сторінки.

Можна виділити такі переваги архітектури:

- висока продуктивність і масштабованість;
- чітке розділення клієнтської та серверної логіки;
- простота розширення функціональності;
- гнучкість інтеграції з зовнішніми API;
- безпечна аутентифікація через JWT;

- підтримка реального часу (через можливість інтеграції з Socket.io).

Таким чином, запропонована архітектура поєднує сучасні технології веброзробки – React, Node.js, та MySQL, – що дозволяє створити ефективну, надійну й масштабовану систему управління проєктами, орієнтовану на командну роботу та підтримку гнучких методологій розробки програмного забезпечення.

3.2. Діаграма прецедентів системи

Діаграма прецедентів є однією з основних складових моделювання поведінки програмної системи, яка відображає взаємодію користувачів (акторів) із системою та перелік основних сценаріїв використання. На ній графічно показано, які функції доступні різним типам користувачів у межах вебсистеми керування проєктами та командної роботи [37, 38].

На рисунку 3.1 подано діаграму прецедентів системи, що описує два основні типи користувачів – звичайного користувача (User) та власника команди (Team Owner). Кожен із них має набір дій (use cases), які відображають функціональні можливості відповідно до їхніх ролей у системі.

- Актор “User” (користувач).

Звичайний користувач системи взаємодіє з інтерфейсом для виконання основних операцій, пов’язаних із роботою над проєктами, командами та завданнями.

До основних сценаріїв використання належать:

- Login (вхід у систему) – автентифікація користувача з використанням облікових даних для отримання доступу до функціональності системи;
- Register (реєстрація) – створення нового облікового запису;
- View Profile (перегляд профілю) – відображення інформації про користувача;
- Edit Profile (редагування профілю) – зміна персональних даних;

- Create Team (створення команди) – створення нової команди для спільної роботи;
- View Team Page (перегляд сторінки команди) – перегляд інформації про команду, її учасників і поточні проєкти;
- Chat (спілкування в чаті) – обмін повідомленнями з іншими членами команди;
- Delete Message from Chat (видалення повідомлення) – можливість видалення власних повідомлень;
- Take Tasks (взяти завдання) – прийняття завдання на виконання;
- View Assigned Tasks (перегляд закріплених завдань) – перегляд переліку завдань, призначених користувачу;
- View Gantt Chart for Projects (перегляд діаграми Ґанта для проєктів) – перегляд часових меж виконання проєктів;
- View Gantt Chart for Tasks (перегляд діаграми Ґанта для завдань) – аналіз стану виконання завдань у межах проєктів.

- Актор “Team Owner” (власник команди).

Власник команди є розширеною роллю звичайного користувача і має додаткові адміністративні можливості.

До основних дій цього актора належать:

- Edit Team Members (редагування учасників команди) – додавання, видалення або зміна ролей членів команди;
- Add Tasks (додавання завдань) – створення нових завдань у межах проєктів;
- Edit Project Deadline (редагування термінів виконання) – зміна кінцевих строків реалізації проєктів;
- Edit Project Name (редагування назви проєкту) – коригування ідентифікаційних даних проєкту;
- Clear Chat (очищення чату) – повне видалення історії повідомлень команди.

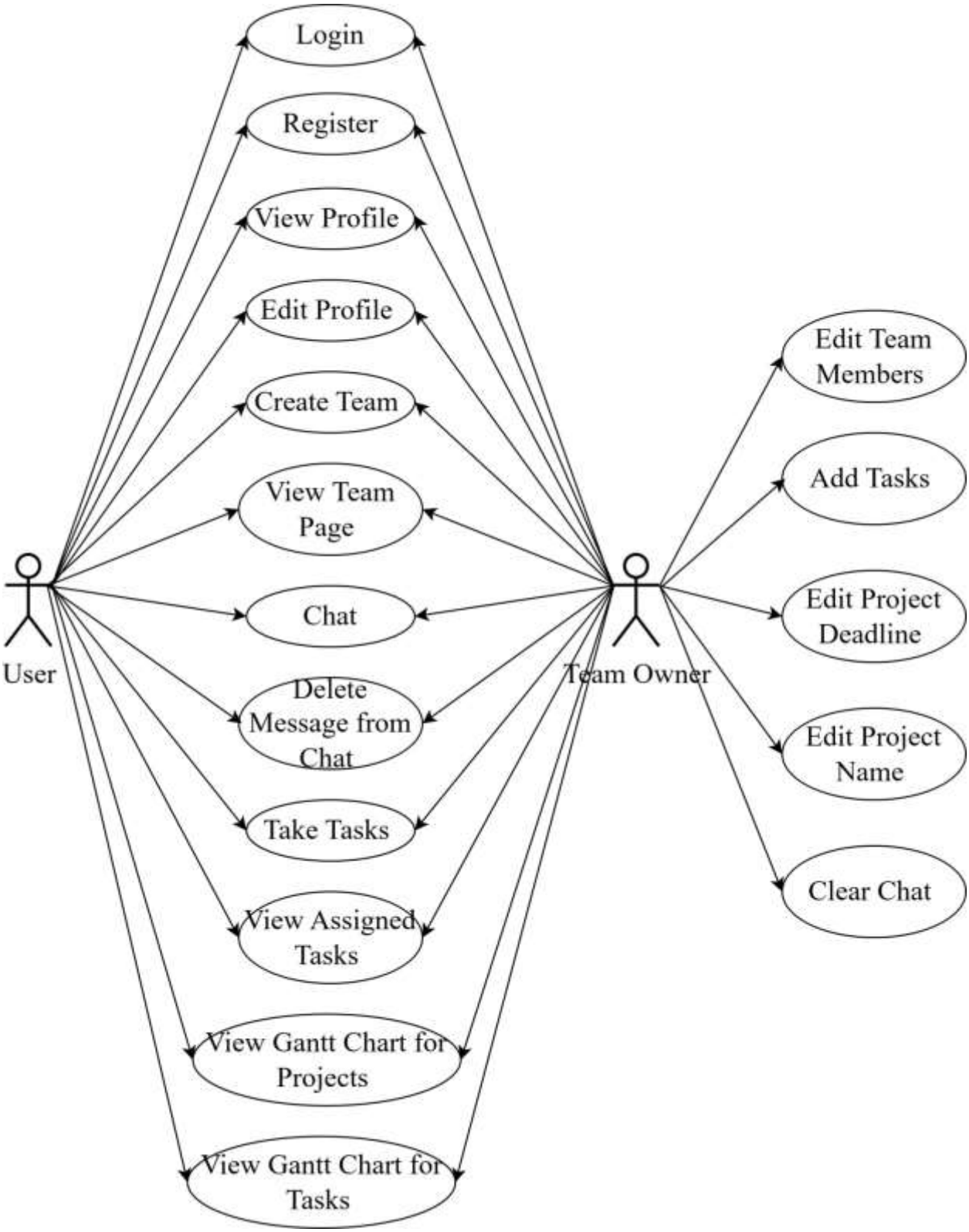

Рис. 3.1. Діаграма прецедентів системи

3.3. Модель бази даних

База даних системи розроблена з використанням системи керування базами даних MySQL, яка є однією з найпоширеніших СУБД завдяки високій швидкодії, надійності та підтримці складних запитів. Модель бази даних створено за допомогою MySQL Workbench, який дозволяє виконати зворотний інжиніринг, побудувати діаграму зв'язків між таблицями (ERD-діаграму) та забезпечити узгодженість логічного і фізичного рівнів моделі.

ERD-діаграма моделі бази даних подана на рис. 3.2. Вона відображає структуру даних та логічні зв'язки між сутностями системи управління проєктами.

Діаграма складається з семи основних сутностей:

1) teams – зберігає інформацію про команди. Основні атрибути:

- id – унікальний ідентифікатор команди (первинний ключ);
- teamName – назва команди;
- userId – ідентифікатори користувачів, які належать до команди;

2) projectdeadlines – містить дані про кінцеві терміни проєктів. Атрибути:

- id – унікальний ідентифікатор запису;
- teamId – зовнішній ключ, що пов'язує таблицю з teams;
- deadlineDate – дата завершення проєкту.

3) taskdeadlines – описує кінцеві терміни для окремих завдань:

- id, taskId, deadlineDate – відповідно ідентифікатор запису, ідентифікатор завдання та дата дедлайну;

4) tasks – основна таблиця для зберігання завдань:

- id – унікальний ідентифікатор завдання;
- taskName – назва або короткий опис завдання;
- teamId – зовнішній ключ, що вказує на команду-виконавця;
- userId – зовнішній ключ користувача, відповідального за завдання;
- status – поточний стан виконання;

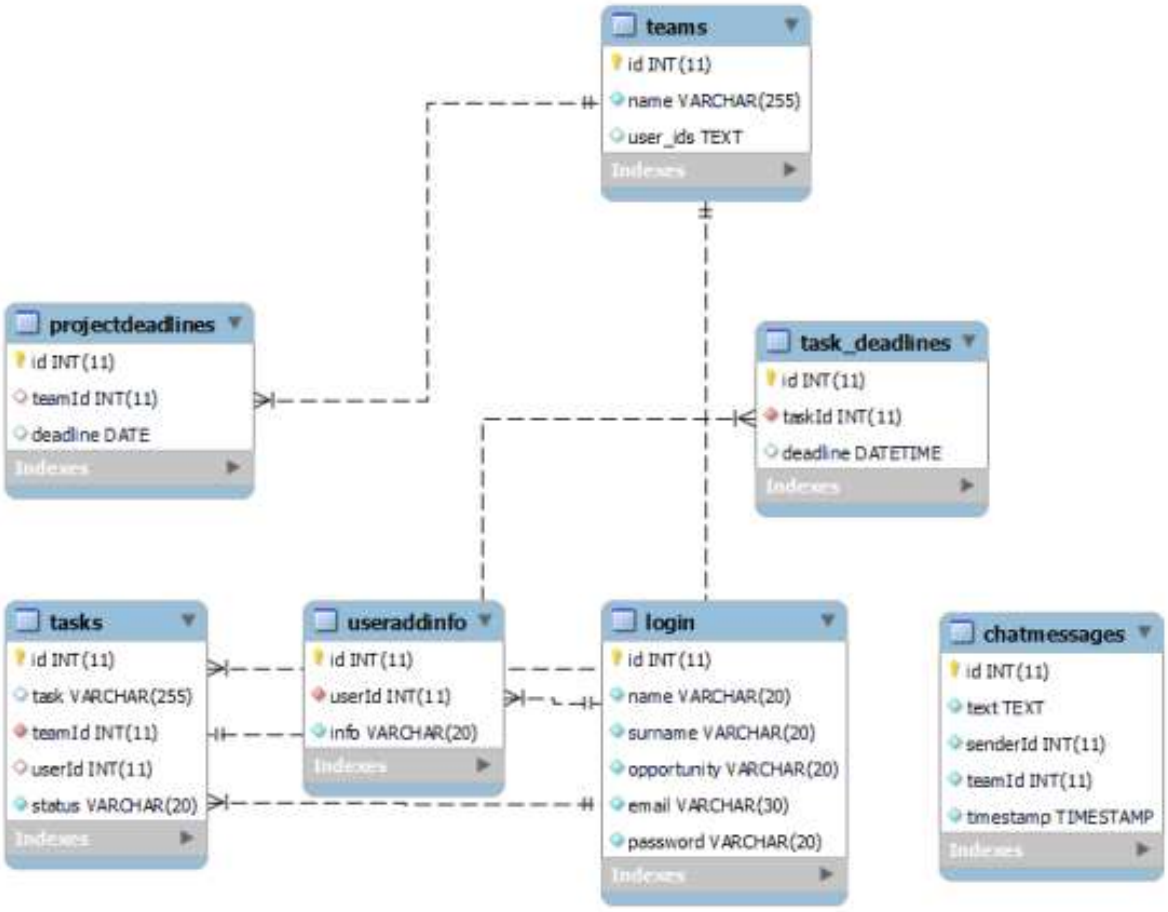

Рис. 3.2. ER-діаграма моделі бази даних системи управління проєктами

5) useraddinfo – містить додаткову інформацію про користувачів:
- id – первинний ключ;
- userInfo – текстова інформація про користувача;
- surname – прізвище користувача;

6) login – забезпечує автентифікацію користувачів у системі:
- id – первинний ключ;
- firstName, lastName – ім’я та прізвище користувача;
- email – електронна адреса;
- password – хешований пароль;
- teamId – зовнішній ключ, що пов’язує користувача з командою;

7) chatmessages – зберігає повідомлення в командному чаті:
- id – унікальний ідентифікатор повідомлення;
- text – вміст повідомлення;

- senderId – ідентифікатор користувача-відправника (зв’язок з таблицею login);
- teamId – ідентифікатор команди, до якої належить чат.

3.4. Програмна реалізація компонентів системи

Розроблена система управління проєктами та командною взаємодією складається з низки окремих компонентів користувацького інтерфейсу, кожен з яких реалізує певний набір функцій, пов’язаних із взаємодією користувача з системою. Усі компоненти створені з використанням бібліотеки React, що забезпечує гнучкість, модульність і швидке оновлення інтерфейсу без необхідності перезавантаження сторінки. Для обміну даними з сервером застосовуються HTTP-запити через бібліотеку Axios, а для побудови діаграм та графічних елементів використано Chart.js та @bitnoi.se/react-scheduler. Система підтримує багатомовність (українську та англійську мови) через динамічне перемикання текстових елементів інтерфейсу.

3.4.1. Компонент авторизації та реєстрації користувачів

Компонент авторизації та реєстрації користувачів реалізовано у вигляді форм із валідацією введених даних. Під час реєстрації (рис. 3.3) користувач заповнює поля ім’я, прізвище, електронну пошту та пароль, після чого інформація надсилається на сервер через POST-запит. Паролі перед збереженням у базі хешуються за допомогою алгоритму bcrypt, що підвищує рівень безпеки системи.

При вході користувача в систему (рис. 3.4) здійснюється перевірка облікових даних, після чого створюється JWT-токен, який зберігається у локальному сховищі браузера (localStorage) для подальшої автентифікації запитів.

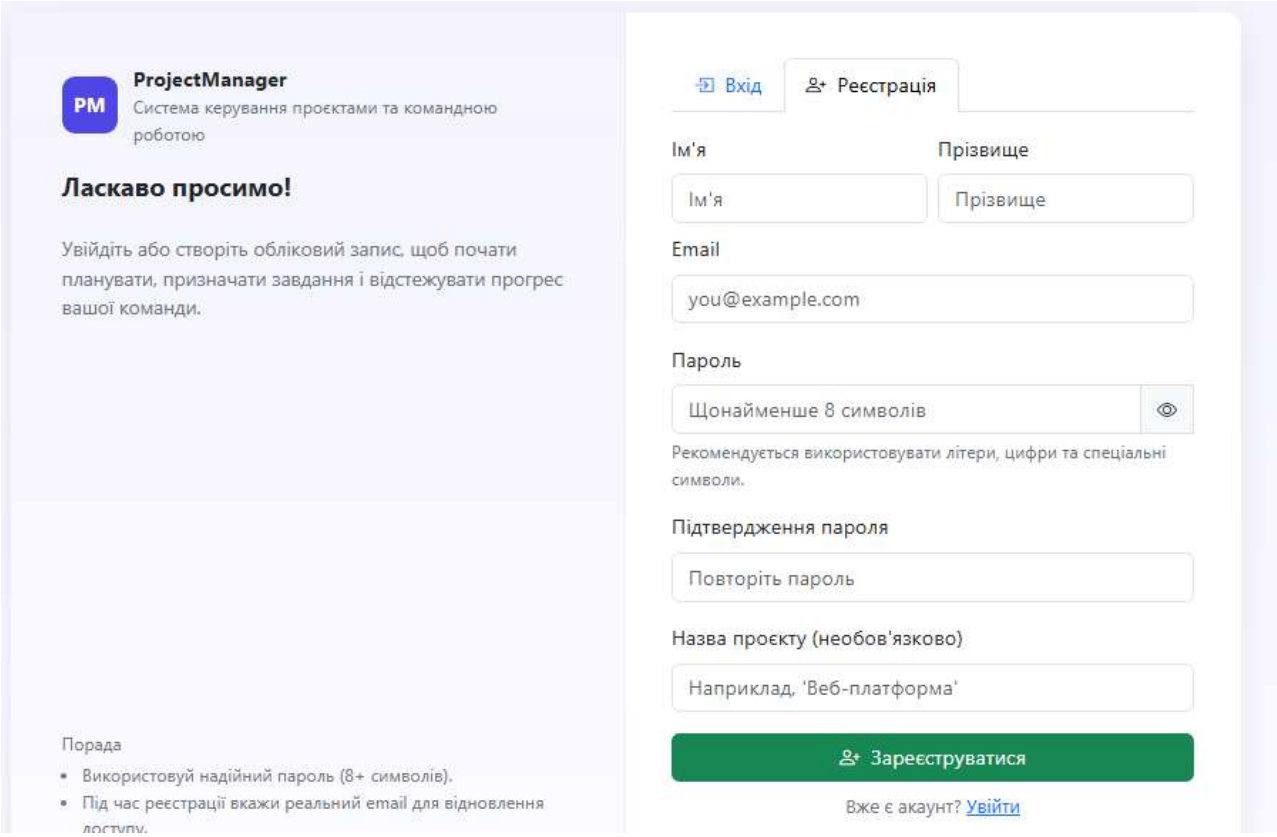

Рис. 3.3. Сторінка реєстрації користувачів

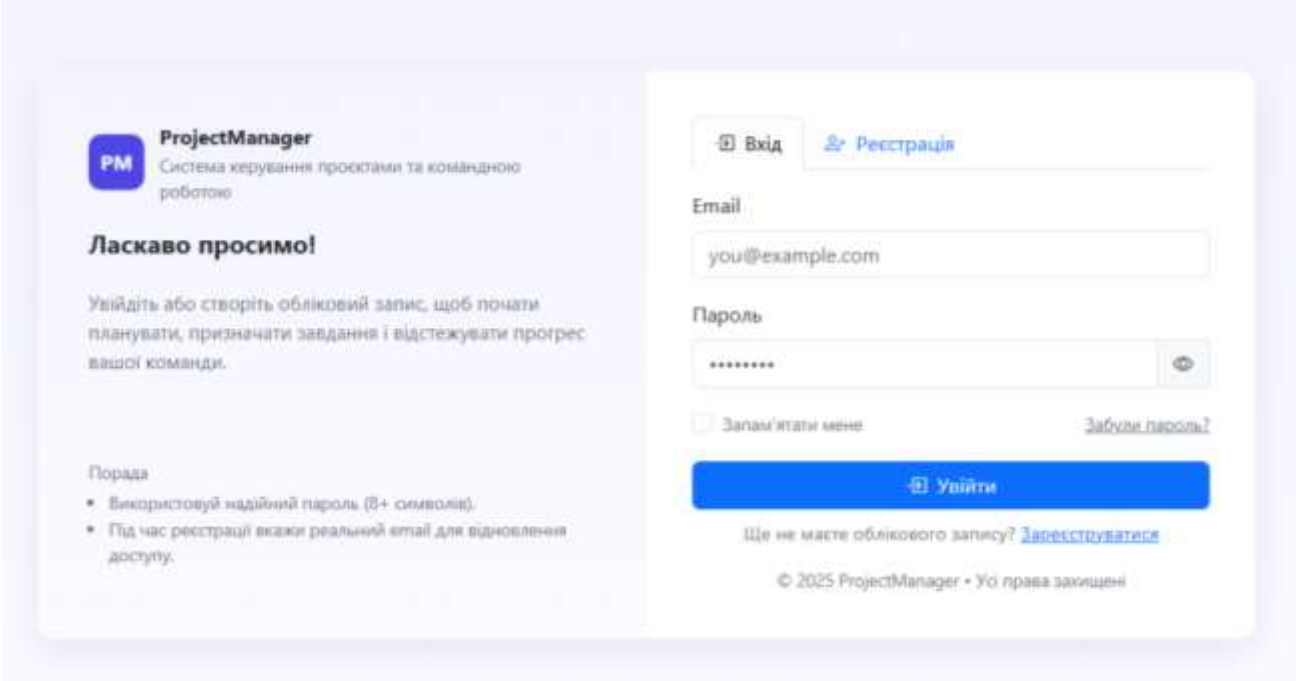

Рис. 3.4. Сторінка авторизації користувачів

3.4.2. Створення та управління командами – компонент TeamDetails

Компонент TeamDetails забезпечує інтерфейс для управління командами користувачів (рис. 3.5). Він реалізує можливість перегляду інформації про команду, редагування параметрів проєкту, керування учасниками, а також видалення команди або виходу з неї. Серед основних функцій компонента – відображення назви проєкту, термінів його виконання, імені організатора команди та переліку учасників. Організатор має можливість редагувати назву проєкту та змінювати кінцевий термін виконання. Додавання нових учасників відбувається через введення адреси електронної пошти, тоді як видалення доступне лише організатору. Компонент підтримує двомовний інтерфейс, а всі операції взаємодіють із сервером у режимі реального часу через API-запити.

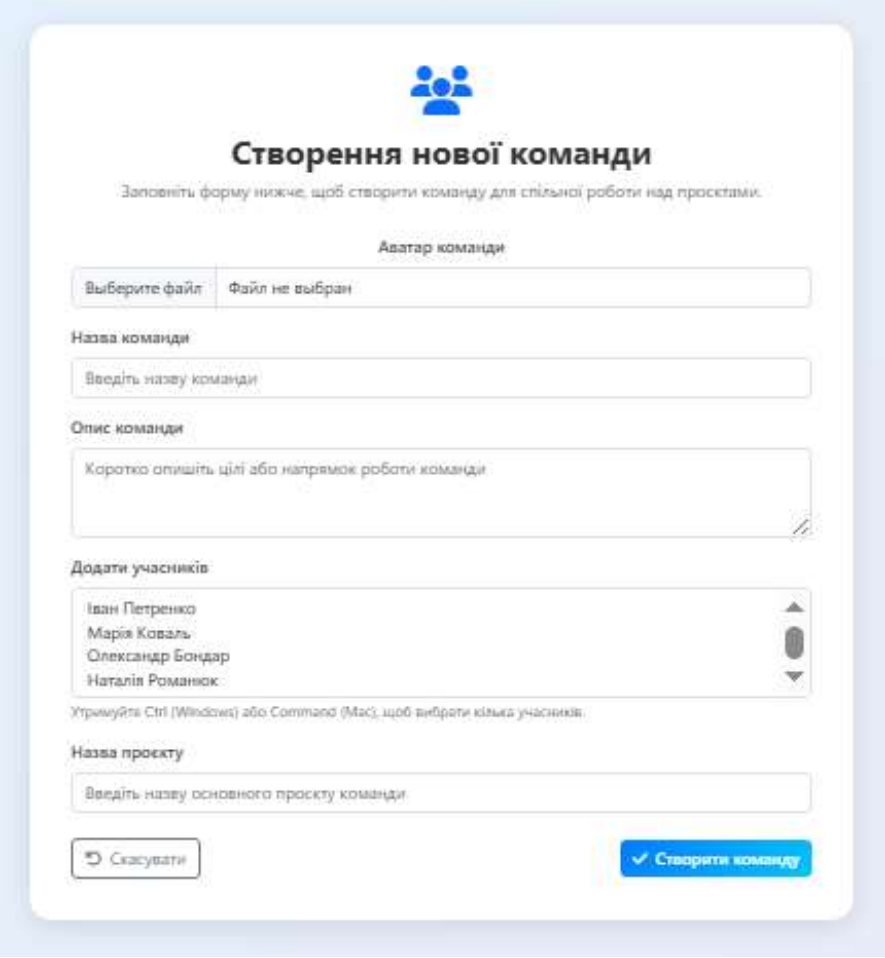

Рис. 3.5. Сторінка створення нової команди

3.4.3. Обмін повідомленнями – компонент Chat

Компонент Chat реалізує функціональність командного чату, що дозволяє учасникам команди обмінюватися повідомленнями в межах поточного проекту (рис. 3.6). Він підтримує надсилання, видалення та очищення історії повідомлень. Кожне повідомлення зберігається на сервері, а відображення в інтерфейсі відбувається в хронологічному порядку. Повідомлення ідентифікуються за відправником, при цьому користувач може видаляти лише власні повідомлення. Для організатора команди передбачено можливість повного очищення чату. Компонент побудовано з використанням React Hooks і забезпечує багатомовну підтримку користувацького інтерфейсу.

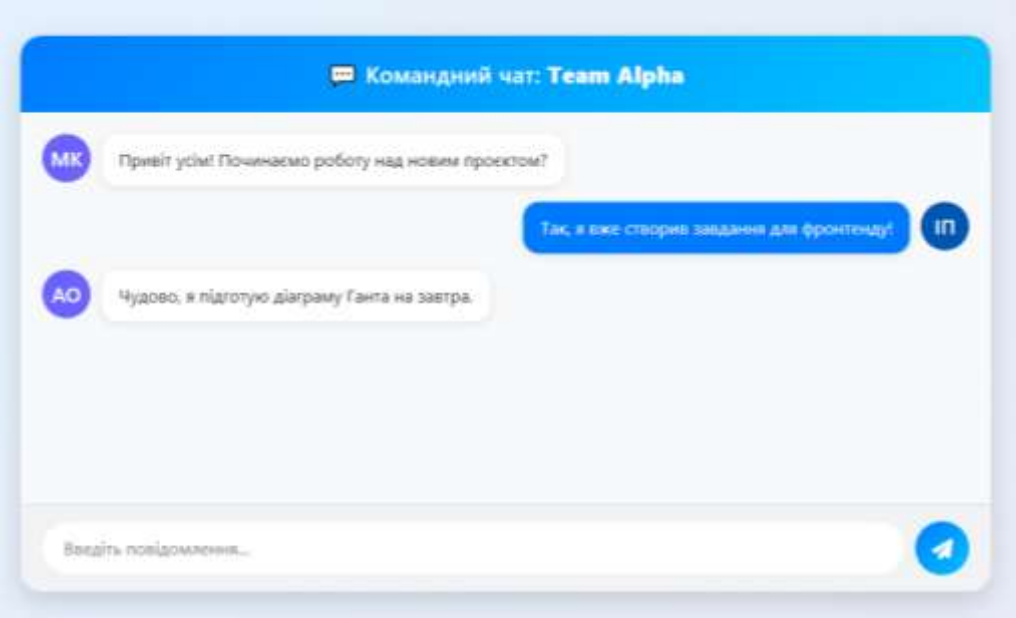

Рис. 3.6. Обмін повідомленнями – компонент Chat

3.4.4. Керування завданнями команд – компонент TaskManagement

Компонент TaskManagement відповідає за управління завданнями у межах команди. Він реалізує функції створення, призначення, скасування призначення, видалення завдань та моніторингу їх виконання. В інтерфейсі

користувачі можуть переглядати список завдань із зазначенням статусу та відповідальних осіб. Організатор має можливість додавати нові завдання, визначати терміни їх виконання та видаляти за потреби. Призначення завдань здійснюється через інтерактивний інтерфейс, а статистика виконання відображається у вигляді кругової діаграми, реалізованої за допомогою бібліотеки Chart.js. Компонент також реалізує підтримку двох мов і взаємодіє із сервером через HTTP-запити.

3.4.5. Відображення та редагування профілю користувача – компонент UserProfile

Компонент UserProfile призначено для відображення та редагування персональних даних користувача (рис. 3.7). Він забезпечує завантаження профільної інформації з бази даних, її оновлення та перевірку коректності введених даних. У режимі редагування користувач може змінювати ім’я, прізвище, електронну адресу, професію та додаткову інформацію. Оновлені дані передаються на сервер за допомогою HTTP PUT-запитів через бібліотеку Axios. Перед збереженням здійснюється валідація даних, зокрема перевірка заповнення обов’язкових полів і унікальності електронної адреси. Компонент підтримує перемикання між режимами перегляду та редагування, а також містить кнопки навігації до головної сторінки та сторінок діаграм Ганта. Додатково реалізовано лічильник символів для поля «Додаткова інформація» з обмеженням у 20 символів.

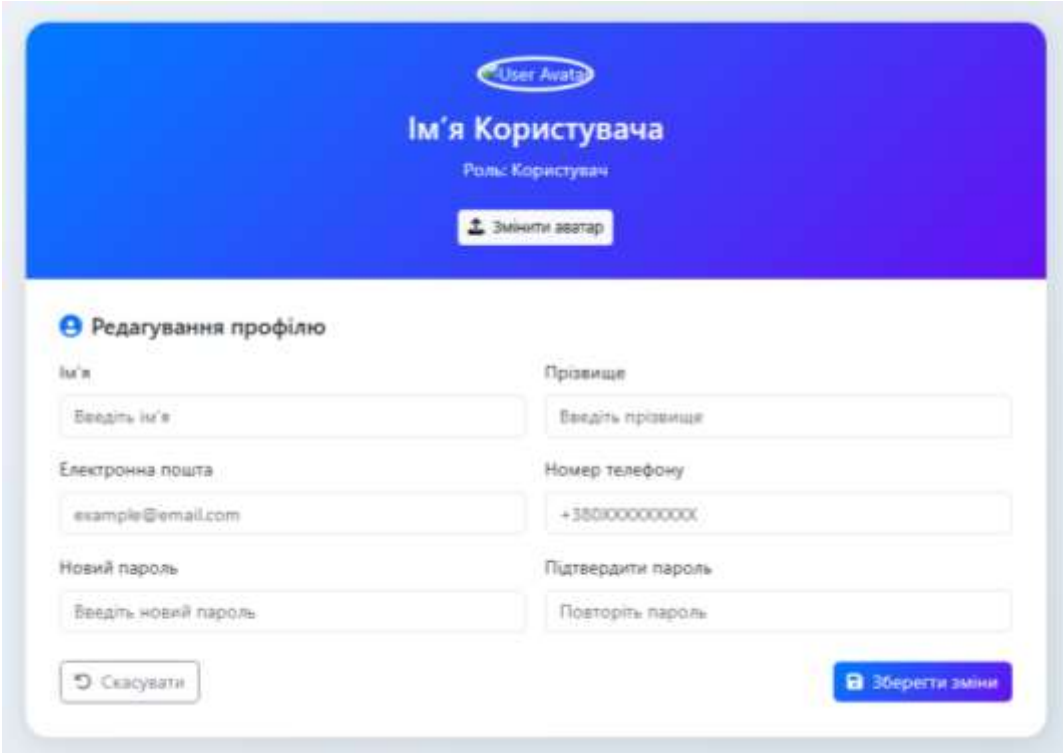

Рис. 3.7. Відображення та редагування профілю користувача

3.4.6. Список завдань користувача – компонент UserTasks

Компонент UserTasks відповідає за відображення переліку завдань, призначених певному користувачу. При монтуванні компонента виконується запит на сервер для отримання актуальних даних про завдання користувача, що авторизується за допомогою JWT-токена. Отримані дані зберігаються у стані React і динамічно оновлюються при зміні ідентифікатора користувача або мови інтерфейсу. У разі відсутності завдань система повідомляє користувача про це, а при їх наявності відображає структурований список із назвою проєкту, текстом завдання та іменем його створювача. Кожен елемент списку є інтерактивним посиланням на сторінку відповідної команди. Компонент підтримує локалізацію інтерфейсу з використанням об’єкта перекладів і зберігає вибір мови в localStorage.

3.4.7. Відображення списку команд користувача – компонент MyTeams

Компонент MyTeams забезпечує відображення списку команд, до яких належить користувач. При ініціалізації виконується послідовність запитів до серверу для отримання ідентифікатора користувача та переліку доступних команд. Система фільтрує ті команди, де поточний користувач присутній у складі учасників, після чого формує список для відображення. Якщо користувач не входить до жодної команди, система відображає відповідне повідомлення. Компонент підтримує створення нової команди, для чого передбачено кнопку переходу до сторінки створення. Реалізована навігація дозволяє швидко повертатися на головну сторінку. Як і в інших компонентах, інтерфейс має багатомовну підтримку з можливістю перемикання мови.

3.4.8. Діаграма Ганта проєктів – компонент GanttChart

Компонент GanttChart реалізує відображення діаграми Ганта для проєктів, пов’язаних із користувачем (рис. 3.8). Для побудови діаграми використано бібліотеку @bitnoi.se/react-scheduler, що забезпечує динамічну візуалізацію часових інтервалів виконання завдань і проєктів. Компонент здійснює запити до серверу для отримання даних про користувача, його команди та кінцеві терміни проєктів. Отримані дані форматуються у структуру, сумісну з планувальником, при цьому для кожного проєкту генерується випадковий колір. Після завершення завантаження інформації компонент відображає діаграму з можливістю масштабування та навігації між елементами. Це дозволяє користувачу візуально оцінювати стан проєктів та контролювати строки виконання.

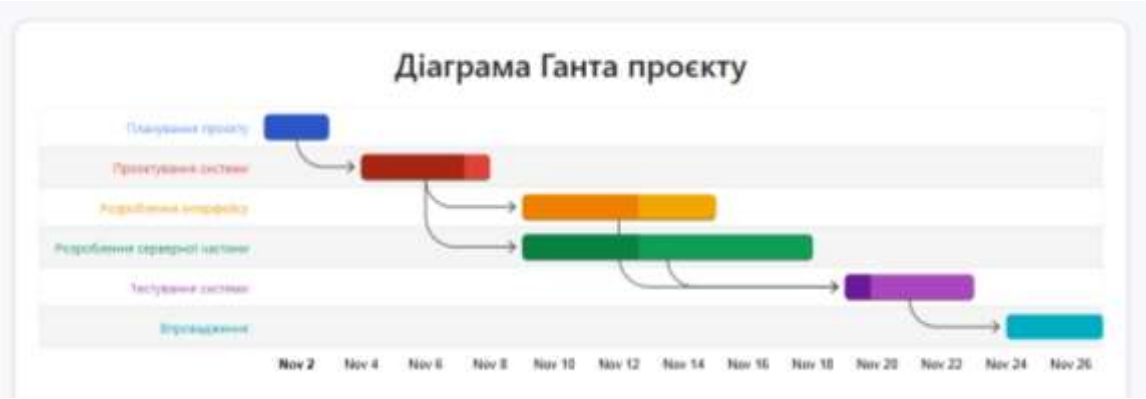

Рис. 3.8. Діаграма Ганта проекту

3.4. Висновки до розділу 3

У третьому розділі здійснено детальний аналіз та практичну реалізацію програмної системи для управління проєктами та командною роботою. Основну увагу приділено архітектурним рішенням, вибору технологічного стеку, моделюванню бази даних, а також створенню окремих компонентів інтерфейсу користувача. Запропонована система побудована на сучасних вебтехнологіях, зокрема використано React для реалізації клієнтської частини, Node.js як серверне середовище виконання, а MySQL — як систему керування базами даних. Такий вибір забезпечує гнучкість, масштабованість і стабільність роботи системи.

Розроблена архітектура системи базується на принципах модульності та багаторівневої взаємодії компонентів. Це дозволило розмежувати логіку користувацького інтерфейсу, бізнес-процеси та доступ до даних, що значно підвищує зручність супроводу та можливість подальшого розширення функціональності.

На основі діаграми прецедентів визначено ключові сценарії взаємодії користувача із системою, що охоплюють процеси авторизації, управління командами, створення завдань, ведення чату та побудову графіків виконання проєктів. Створена модель бази даних оптимізована для зберігання

взаємопов’язаних сутностей з урахуванням цілісності даних і швидкого доступу.

У ході реалізації компонентів системи було розроблено низку ключових модулів:

– компонент авторизації та реєстрації забезпечує безпечну і зручну роботу користувачів із системою, підтримуючи валідацію введених даних та захист від несанкціонованого доступу;

– компонент управління командами дозволяє створювати команди, призначати учасників і керувати їхніми ролями;

– компонент обміну повідомленнями реалізує інтерактивну комунікацію між користувачами, підтримуючи відображення повідомлень у реальному часі;

– компонент керування завданнями відповідає за створення, редагування та контроль виконання задач у межах проєктів;

– компонент профілю користувача забезпечує персоналізацію та можливість редагування персональних даних;

– компонент списку завдань користувача надає швидкий доступ до індивідуальних робочих завдань;

– компонент відображення списку команд структурує інформацію про участь користувача у різних командах;

– компонент діаграми Ганта реалізує візуалізацію етапів виконання проєктів, що дає змогу ефективно відстежувати часові залежності та прогрес.

У підсумку, розроблена система управління проєктами та командною взаємодією демонструє ефективне поєднання сучасних вебтехнологій, продуманої архітектури та гнучких інтерфейсних рішень. Її структура дозволяє легко масштабувати функціонал, адаптувати систему під потреби різних організацій і забезпечує високу якість керування життєвим циклом програмних проєктів.

ВИСНОВКИ

У кваліфікаційній магістерській роботі було проведено дослідження впливу гнучких методологій управління проєктами на ефективність процесу розроблення програмного забезпечення, а також реалізовано вебсистему для управління проєктами та командною взаємодією, побудовану з урахуванням принципів Agile-підходів.

На основі проведеного аналізу теоретичних засад гнучких методологій (Scrum, Kanban, Lean, Extreme Programming тощо) встановлено, що їх впровадження забезпечує підвищення адаптивності команди до змін, скорочення часу розроблення продукту, зростання якості коду та рівня задоволеності замовника. Особлива увага приділялась порівнянню класичних та гнучких підходів, що дозволило визначити ключові переваги останніх – ітеративність, прозорість процесів, постійний зворотний зв’язок та орієнтація на цінність продукту.

У другому розділі досліджено процес управління проєктом у межах гнучких методологій, розроблено модель етапів планування, створено графік виконання робіт та побудовано діаграму Ганта. Це дало змогу продемонструвати практичні аспекти планування, контролю виконання завдань і розподілу ролей між учасниками команди. Здійснено аналіз взаємозв’язку між правильним плануванням та ефективністю реалізації проєкту.

У практичній частині виконано проєктування та програмну реалізацію вебсистеми для управління програмними проєктами, яка містить модулі: авторизації та реєстрації користувачів, управління профілем, створення та керування командами, інтегрований чат для комунікації учасників, а також модуль відображення діаграми Ганта для моніторингу процесу виконання завдань. Реалізація здійснювалась із використанням сучасного технологічного стеку – React для побудови клієнтської частини, Node.js для серверної логіки, MySQL як системи керування базами даних, а також

HTML5, CSS3, JavaScript і фреймворку Bootstrap для створення адаптивного та зручного інтерфейсу користувача. Крім того, застосовано бібліотеки для візуалізації даних і побудови діаграм, що дало змогу реалізувати інструмент відображення графіка виконання завдань і оцінки критичного шляху проєкту.

Система забезпечує інтуїтивно зрозумілий інтерфейс, підтримує масштабованість і можливість подальшого розширення функціоналу. Вона відповідає сучасним вимогам до веборієнтованих рішень, поєднуючи гнучкість архітектури та високу продуктивність.

Проведене дослідження має практичну цінність, оскільки розроблена система може бути використана як прототип для впровадження в реальні IT-команди, що працюють за методологіями Agile. У подальшому робота може бути розширена шляхом інтеграції з зовнішніми системами управління завданнями, додаванням аналітичних модулів та засобів автоматизованого контролю ефективності роботи команди.

ПЕРЕЛІК ПОСИЛАНЬ

1. Лавріщева К. М. Програмна інженерія. Київ, 2008. 319 с. ISBN 978–966–02–5052–9.

2. Sommerville I. Software Engineering / I. Sommerville. London : Pearson, 2015. 816 p.

3. Richards M., Ford N. Fundamentals of Software Architecture: An Engineering Approach / M. Richards, N. Ford. Sebastopol : O'Reilly Media, 2020. 432 p.

4. Дегтярьова Л. М., Гроза П. М., Сомов С. В. Технології розробки програмного забезпечення : навч. посіб. для студентів спец. 123 «Комп’ютерна інженерія». Полтава : ПолтНТУ, 2017. 218 с.

5. Ковалюк Д. О. Технології розроблення програмного забезпечення: практикум [Електронний ресурс] : навч. посіб. для студ. спец. 151 «Автоматизація та комп’ютерно-інтегровані технології», освітньо-професійна програма «Технічні та програмні засоби автоматизації» / уклад. Д. О. Ковалюк. Київ : КПІ ім. Ігоря Сікорського, 2022. 55 с.

6. Марченко О. І., Вітковська І. І. Компоненти програмної інженерії. Частина 1. Вступ до програмної інженерії : лабораторний практикум [Електронний ресурс] : навч. посіб. для студ. освіт. програми «Інженерія програмного забезпечення інформаційних систем» спец. 121 «Інженерія програмного забезпечення» / О. І. Марченко, І. І. Вітковська. Київ : КПІ ім. Ігоря Сікорського, 2022. 50 с.

7. Вітковська І. І., Крамар Ю. М. Основи програмування. Частина 2. Модульне програмування : лабораторний практикум [Електронний ресурс] : навч. посіб. для студ. спец. 121 «Інженерія програмного забезпечення» / КПІ ім. Ігоря Сікорського. Київ : КПІ ім. Ігоря Сікорського, 2025. 108 с.

8. Трофименко О. Г., Манаков С. Ю., Ларін Д. Г. Основи програмної інженерії : навч.-метод. посіб. Одеса : Фенікс, 2022. 197 с.

9. Бородкіна І. Л., Бородкін Г. О. Інженерія програмного забезпечення

: посіб. для студентів вищих навч. закладів / Мво освіти і науки України, Нац. ун-т біоресурсів та природокористування України. Київ, 2018. 251 с.

10. Корнієнко Б. Я. Компоненти програмної інженерії. Архітектура програмного забезпечення : лабораторний практикум [Електронний ресурс] : навч. посіб. для здобувачів ступеня бакалавра спец. 121 «Інженерія програмного забезпечення» / КПІ ім. Ігоря Сікорського. Київ : КПІ ім. Ігоря Сікорського, 2023. 88 с.

11. Чопоров С. В., Чопорова О. В., Кривохата А. Г. Основи програмної інженерії : навч. посіб. до лаб. занять для здобувачів ступеня вищої освіти бакалавра спец. «Інженерія програмного забезпечення» освіт.-проф. програми «Програмна інженерія». Запоріжжя : ЗНУ, 2022. 112 с.

12. Левус Є. В., Мельник Н. Б. Вступ до інженерії програмного забезпечення : навч. посіб. Львів : Видавництво Львівської політехніки, 2018. 248 с.

13. Левус Є. В., Марусенкова Т. А., Нитребич О. О. Життєвий цикл програмного забезпечення : навч. посіб. Львів : Видавництво «Львівська політехніка», 2017. 208 с.

14. Agile-маніфест розробки програмного забезпечення [Електронний ресурс] : URL: https://agilemanifesto.org/iso/uk/manifesto.html

15. A guide to the Project Management Body of Knowledge. PMBOK guide SIXTH EDITION. USA : Project Management Institute, 2017.

16. Близнюкова І. О., Семко С. Г., Кійко С. Г. Огляд сучасних методологій управління командами ІТ-проєктів. Управління розвитком складних систем. Київ, 2020. № 43. С. 60–66. https://doi.org/10.32347/2412-9933.2020.43.60-66

17. Колянко О. В., Озимок Г. В. Використання жорсткої “Waterfall” та гнучкої “Agile” моделей управління проєктами. Вісник Львівського торговельно-економічного університету. Економічні науки. Львів, 2017. Вип. 52. С. 177–182.

18. Вавіленкова А. І. Аналіз гнучких методологій розробки

програмного забезпечення для реалізації у командних проєктах. Вісник НТУ «ХПІ». Харків, 2021. № 1(7). С. 39–46. https://doi.org/10.20998/2413-4295.2021.01.06

19. Кім О. О., Козлова В. В. Перспективи застосування методології Agile-менеджменту в управлінні ІТ-проєктами. Соціальна економіка. Харків, 2019. № 58. С. 95–99. https://doi.org/10.26565/2524-2547-2019-58-12

20. Кордунова Ю. С., Придатко О. В., Смотр О. О. Переваги використання Agile-методології під час розробки програмного забезпечення в умовах сучасного ринку. Інформаційна безпека та інформаційні технології : зб. наук. праць IV Всеукр. наук.-практ. конф. молодих учених, студентів і курсантів. Львів, 2020. С. 206–207.

21. Кордунова Ю. С., Смотр О. О. Сенс Agile-маніфесту для сучасного проєкт-менеджменту. Проблеми та перспективи розвитку системи безпеки життєдіяльності: зб. наук. праць XVI Міжнар. наук.-практ. конф. молодих вчених, курсантів та студентів. Львів : ЛДУ БЖД, 2021. С. 247-248.

22. Приймак В. Гнучкі моделі управління командною роботою інжинірингових проєктів. Вісник Київського національного університету імені Тараса Шевченка. Економіка. Київ, 2019. № 6 (207). С. 21-27. https://doi.org/10.17721/1728-2667.2019.207-6/3

23. Якубенко І. М. Agile-менеджмент, як дієве управління проєктами для цілеспрямованих команд. Економіка. Менеджмент. Бізнес. 2017. № 4(22). С. 167–172.

24. Муравецький С. А., Крамський С. О. Планування процесів забезпечення якості у великих та географічно розподілених гібридних ІТ-проєктах. Вісник НТУ «ХПІ». Харків, 2016. № 1(1173). С. 106–109. https://doi.org/10.20998/2413-3000.2016.1173.21

25. Асєєва А. В., Кулаковська І. В. Аналіз проблем вибору технології для розробки програмного забезпечення. Комп’ютерно-інтегровані технології: освіта, наука, виробництво. Луцьк, 2019. № 37. С. 10–18.

26. Бушуєв С.Д., Бушуєва В. Б., Бойко О. О. Agile-трансформація

підходів в управлінні будівельними проєктами, фазах ініціалізації та проєктування. Управління розвитком складних систем. Київ, 2020. № 41. С. 14-20. https://doi.org/10.32347/2412-9933.2020.41.15-20

27. Ґидроец М. О., Гришанова Л. И. Методології розробки програмного продукту. Системний аналіз і логістика. Санкт-Петербург, 2020. № 4 (26). С. 45-52. https://doi.org/10.31799/2007-5687-2020-4-45-53

28. Barraood S. O., Mohd H., Baharom F. A comparison study of software testing activities in Agile methods. Knowledge Management International Conference (KMICe) Virtual Conference. Malaysia, 2021. С. 130–137.

29. Семенов С. Г., Халіфе Кассем, Захарченко М. М. Усовершенствований спосіб масштабування гнучкої методології розробки програмного забезпечення. Вісник НТУ «ХПІ». Харків, 2017. Т. 1, № 1. С. 79-84. https://doi.org/10.20998/2522-9052.2017.1.15

30. Шапошнікова О. П., Кірвас В. В. Застосування методології Agile в практиці проєктного навчання при підготовці ІТ-спеціалістів. Системи обробки інформації. Харків, 2020. № 4(163). С. 94-100. https://doi.org/10.30748/soi.2020.163.10

31. Козир І. С. Фактори впровадження Agile-менеджменту в практику управління. I International Scientific and Practical Conference «Problemas y perspectivas de la aplicación de la investigación científica innovadora». Кембридж, 2021. Т. 1. С. 78-79. https://doi.org/10.36074/logos-19.03.2021.v1.26

32. Придатко О. В., Солотвінський І. В., Кокотко І. Я., Івановський М. Я. Модель портфельного управління проєктами розвитку регіональних систем безпеки життєдіяльності. Управління розвитком складних систем. Київ, 2018. № 36. С. 42–50.

33. Кордунова Ю. С., Смотр О. О. Визначення ефективності використання Agile-методології в сучасних організаціях. Проблеми та перспективи забезпечення цивільного захисту: матеріали міжнародної науково-практичної конференції молодих учених. Харків : НУЦЗУ, 2021. С. 166.

34. Islam G., Stoner T. A case study of agile software development for safety-critical systems projects. Reliability Engineering & System Safety. Vol. 200. 2020. https://doi.org/10.1016/j.ress.2020.106954

35. Stellman A., Greene J. Learning Agile: Understanding Scrum, XP, Lean, and Kanban. 1-st ed. USA : O'Reilly Media, 2013. 420 p.

36. Stioca M., Ghlic-Micu B., Mircea M., Uscatu C. Analyzing Agile Development – from Waterfall Style to Scrumban. Informatica Economică. 2016. № 4. С. 5-14. https://doi.org/10.12948/issn-1453-1305/20.4.2016.01

37. Cole R., Scotcher E. Brilliant Agile Project Management: A Practical Guide to Using Agile, Scrum and Kanban. Edinburgh : Pearson, 2015. 187 p.

38. Papadopoulos G. Moving from traditional to agile software development methodologies also on large, distributed projects. Procedia – Social and Behavioral Sciences. 2015. № 175. С. 455-463. https://doi.org/10.1016/j.sbspro.2015.01.1223