Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Інформаційних управляючих систем
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження гнучких гібридних моделей життєвого циклу ІТ-проєкту

(тема)

Виконав:
здобувач 2 року навчання,
групи УПГІТм-23-2
Максим Мартиненко
(власне ім'я, прізвище)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Управління проєктами в галузі ІТ
(повна назва освітньої програми)

Керівник: _проф. каф. ІУС Максим ЄВЛАНОВ
(посада, власне ім'я, прізвище)

Допускається до захисту

Зав. кафедри ІУС _____ Костянтин ПЕТРОВ
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Інформаційних управляючих систем

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Управління проєктами в галузі інформаційних технологій
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)

" 21 " квітня 20 25 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Мартиненку Максиму Віталійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження гнучких гібридних моделей життєвого циклу IT-проєкту

затверджена наказом по університету від " 28 " березня 2025 р. № 235Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії " 02 " червня 2025 р.

3. Вихідні дані до роботи Agile-маніфест, моделі життєвого циклу, гнучкі методології, типи IT-проєктів

4. Перелік питань, що потрібно опрацювати у роботі Класифікація моделей життєвого циклу, аналіз гнучких методологій, дослідження моделі Crystal, порівняння типів IT-проєктів, визначення критеріїв вибору моделей, розробка рекомендацій з адаптації.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Змістовна постановка задачі призначення пакетів робіт командам виконавців ІТ-проєкту	21.04.2025 - 25.04.2025	Виконано
2	Аналіз літературних джерел і огляд предметної області	26.04.2025 - 28.04.2025	Виконано
3	Дослідження гнучких методологій Agile	29.04.2025 - 03.05.2025	Виконано
4	Аналіз різновидів ІТ-проєктів та вимог до управління ними	04.05.2025 - 08.05.2025	Виконано
5	Порівняння гібридних моделей життєвого циклу з урахуванням типів проєктів	09.05.2025 - 13.05.2025	Виконано
6	Розробка критеріїв вибору гнучкої моделі під конкретний тип проєкту	14.05.2025 - 18.05.2025	Виконано
7	Адаптація методології Crystal під різні типи ІТ-проєктів	19.05.2025 - 26.05.2025	Виконано
8	Апробація результатів, узагальнення висновків і оформлення роботи	27.05.2025 - 29.05.2025	Виконано
9	Оформлення пояснювальної записки	30.05.2025 - 02.06.2025	Виконано
10	Оформлення графічної частини та презентаційних матеріалів захисту	04.06.2025	Виконано
11	Проведення передзахисту кваліфікаційної роботи	05.06.2025	Виконано
12	Захист кваліфікаційної роботи	06.06.2025	

Дата видачі завдання 21 квітня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ проф. каф. ІУС Максим ЄВЛАНОВ
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 122 с., 7 рис., 17 табл., 1 дод., 18 джерел, 1 формула

ГНУЧКЕ УПРАВЛІННЯ, ГІБРИДНА МОДЕЛЬ, ЖИТТЄВИЙ ЦИКЛ, AGILE, ІТ-ПРОЄКТИ, КЛАСИФІКАЦІЯ, АДАПТАЦІЯ, МЕТОДОЛОГІЇ.

Об’єктом дослідження кваліфікаційної роботи є процес управління життєвим циклом ІТ-проєктів.

Мета роботи - аналіз існуючих гнучких гібридних моделей життєвого циклу та розробка рекомендацій з адаптації методології Crystal для ефективного управління різними типами ІТ-проєктів (продуктовими, аутсорсинговими, R&D, підтримки та внутрішніми корпоративними).

У роботі здійснено комплексний огляд сучасних моделей життєвого циклу ІТ-проєктів, зокрема каскадної, спіральної, ітеративної, інкрементної, прототипної, а також гнучких методів Scrum, Kanban, XP, Lean, Scrumban, Crystal. Проведено порівняльний аналіз гнучких гібридних підходів та типів ІТ-проєктів з визначенням критеріїв вибору відповідної моделі управління.

Особливу увагу приділено методології Crystal, як одній з найбільш адаптивних гнучких систем. Розроблено рекомендації з її адаптації до кожного з п’яти типів ІТ-проєктів. Проведено апробацію розроблених підходів на прикладі реального ІТ-проєкту з оцінкою результатів впровадження.

Наукова новизна полягає в практичній адаптації гнучкої методології Crystal до різних умов реалізації проєктів в ІТ-сфері з урахуванням їх специфіки, критичності, розміру команди та рівня змінності вимог.

Практичне значення роботи полягає у можливості використання запропонованих підходів при плануванні, організації та контролі ІТ-проєктів у реальному бізнес-середовищі.

ABSTRACT

Master’s thesis: 122 pages, 7 figures, 17 tables, 1 appendices, 18 sources, 1 formula

FLEXIBLE MANAGEMENT, HYBRID MODEL, LIFE CYCLE, AGILE, IT PROJECTS, CLASSIFICATION, ADAPTATION, METHODOLOGY.

The object of research of the qualification work is the process of IT project lifecycle management.

The aim of the work is to analyse the existing flexible hybrid life cycle models and to develop recommendations for adapting Crystal methodology for effective management of different types of IT projects (product, outsourcing, R&D, support and in-house).

The paper presents a comprehensive review of modern IT project lifecycle models, including cascading, spiral, iterative, incremental, prototype, and agile methods such as Scrum, Kanban, XP, Lean, Scrumban, and Crystal. A comparative analysis of agile hybrid approaches and types of IT projects is carried out, identifying criteria for selecting a suitable management model.

Special attention is paid to the Crystal methodology as one of the most adaptive agile systems. The authors have developed recommendations for its adaptation to each of the five types of IT projects. Results. Approbation of the developed approaches on the example of a real IT-project with evaluation of the implementation results.

The scientific novelty of the article lies in the practical adaptation of the Crystal agile methodology to different conditions of project implementation in the IT industry, taking into account their specificity, criticality, team size and the level of variability of requirements.

The practical significance of the work lies in the possibility of using the proposed approaches in planning, organising and controlling.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ПЗ - програмне забезпечення

ЖЦ - життєвий цикл

XP - Extreme Programming

TDD - Test-Driven Development

ВСТУП

У сучасному світі стрімких змін і розвитку інформаційних технологій ефективне управління IT-проєктами є критичним чинником успіху бізнесу. Зі зростанням складності проєктів і динамікою вимог замовників традиційні підходи з фіксованими планами дедалі частіше виявляються недостатніми. Це зумовлює потребу у впровадженні більш гнучких методів, здатних адаптуватися до змін і забезпечувати ефективну взаємодію всіх учасників.

Одним із таких підходів є гнучкі (Agile) методології, що дозволяють командам швидко реагувати на зміни, підтримувати тісну комунікацію із замовником і знижувати ризики. До найпоширеніших належать Scrum, Kanban, Lean, Extreme Programming. Вони базуються на ітеративному підході до розробки продукту, що дає змогу гнучко підходити до реалізації вимог. Проте ці методології мають і обмеження - зокрема, труднощі з масштабуванням або координацією великих команд.

У відповідь на ці виклики дедалі частіше застосовуються гібридні моделі управління проєктами, які поєднують елементи як гнучких, так і традиційних підходів. Такий підхід дозволяє зберігати гнучкість і швидкість прийняття рішень, водночас забезпечуючи структурованість і контроль, притаманні класичному проєктному менеджменту.

Гібридні моделі дають змогу обирати найбільш доцільні інструменти залежно від конкретного етапу чи особливостей проєкту. Наприклад, жорстке планування може застосовуватися на початкових етапах, тоді як подальші фази можуть реалізовуватись за допомогою гнучких підходів. Це особливо важливо для великих IT-проєктів, де необхідна координація між кількома командами й використання різних технологій.

Дослідження таких гібридних моделей є надзвичайно актуальним у контексті постійних технологічних змін. Важливо не лише аналізувати їхні переваги та недоліки, а й оцінювати ефективність у порівнянні з традиційними

методами, а також вивчати приклади успішного впровадження в реальних умовах.

Метою цього дослідження є комплексний аналіз гнучких гібридних моделей життєвого циклу ІТ-проєктів, їх ефективності та потенціалу адаптації до різних проєктних умов. Особлива увага приділяється поєднанню планування з адаптивністю, що забезпечує високий рівень контролю і водночас гнучкість при реалізації завдань.

Кваліфікаційна робота виконується згідно з державними стандартами та методичними вказівками щодо розробки та оформлення кваліфікаційної роботи другого (магістерського) рівня вищої освіти [1].

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВА ЗАДАЧІ

1.1 Гібридні моделі життєвого циклу

1.1.1 Поняття життєвого циклу програмного проєкту

Кожен проєкт має свій унікальний процес виконання, який розтягнутий у часі та визначається як життєвий цикл. Цей цикл охоплює всі етапи від зародження ідеї до її повної реалізації, включаючи чітко окреслені початок і завершення. Саме завдяки життєвому циклу проєкту забезпечується логічна послідовність дій, яка дозволяє ефективно управляти ресурсами, ризиками та досягати поставлених цілей.

Як уже зазначалося, кінцевим результатом будь-якого проєкту є унікальний продукт або послуга, що має свою особливу цінність. Оскільки кожен продукт створюється з урахуванням специфічних вимог, процес його розробки також набуває певної унікальності. Однак, незважаючи на це, менеджер проєкту не розробляє життєвий цикл щоразу з нуля. Натомість він спирається на перевірені практики та загальноприйняті методології, які допомагають структурувати робочий процес, забезпечити його послідовність і гнучкість. Водночас ці підходи не є жорстко фіксованими - вони можуть і повинні адаптуватися відповідно до специфіки конкретного проєкту, його масштабів, галузі та вимог замовника.

З часом з’явилося та поступово розвивалося багато моделей життєвого циклу проєкту, кожна з яких має свої особливості, сильні та слабкі сторони. Одні моделі орієнтовані на гнучкість і швидке реагування на зміни, інші - на чітку структурованість та детальне планування. Саме тому успішний менеджер проєкту має глибоко розуміти різні підходи, оцінювати їхні переваги та недоліки, щоб обрати оптимальну модель, яка найкраще відповідатиме цілям і умовам конкретного проєкту.

1.1.2 Зв’язок моделі життєвого циклу програмного проєкту і підходів до управління та показників програмних проєктів

Управління програмними проєктами - це складний процес, що вимагає не лише технічних знань, але й уміння організувати роботу команди на всіх етапах проєкту. Одним з основних факторів, що впливають на успішність проєкту, є правильний вибір моделі життєвого циклу розробки програмного забезпечення. Саме модель життєвого циклу визначає, як будуть структуровані етапи розробки та як буде організована робота команди. Вибір цієї моделі напряму впливає на управління проєктом та його кінцеві результати.

Моделі життєвого циклу програмного проєкту описують етапи, через які проходить проєкт - від визначення вимог до підтримки і оновлення продукту після його запуску. Важливою характеристикою кожної моделі є її здатність адаптуватися до змін, адже розробка програмного забезпечення - це процес, що часто потребує коригування на різних етапах. Існують різні моделі життєвого циклу, і кожна з них має свої особливості. Наприклад, водоспадна модель підходить для проєктів, де вимоги є чітко визначеними на початку і не змінюються протягом розробки. Інші моделі, як-от спіральна чи гнучкі методології, дозволяють зберігати гнучкість і адаптуватися до змін, що особливо важливо для складних і швидко змінюваних проєктів.

Зв'язок між вибором моделі життєвого циклу і підходом до управління проєктом є надзвичайно важливим. Якщо проєкт потребує чітко спланованих етапів і контрольованих змін, то традиційні моделі, такі як водоспадна, будуть найбільш підходящими. Вони дозволяють детально планувати кожен етап і знижують ймовірність помилок, оскільки кожен наступний етап будується на основі попереднього. Проте, для проєктів, де зміни вимог можливі навіть після старту, більш підходять гнучкі методології, такі як Scrum або Kanban. Вони

дозволяють адаптуватися до нових умов і змін у вимогах, що є важливим для проєктів з високим рівнем невизначеності.

Підходи до управління проєктами визначають, як буде організовано виконання кожного етапу, хто і як буде контролювати хід роботи та реагувати на зміни. Для традиційних моделей життєвого циклу характерна чітка ієрархія та контроль за виконанням кожного етапу, в той час як у гнучких методах управління велика увага приділяється швидкому зворотному зв'язку і адаптації. Гнучкість тут є ключовою перевагою, оскільки дозволяє змінювати вимоги або коригувати стратегію в процесі розробки, що підвищує ймовірність успіху.

Успішність програмного проєкту часто вимірюється через кілька основних показників, таких як час, бюджет і якість продукту. Ці фактори тісно пов'язані з вибором моделі життєвого циклу та підходу до управління. Наприклад, якщо проєкт завершується вчасно, у межах бюджету і з високою якістю продукту, це свідчить про ефективне управління та правильний вибір моделі. У свою чергу, якщо проєкт затримується або перевищує бюджет, це може свідчити про недоліки в управлінні або невірно обрану модель життєвого циклу.

Таким чином, зв'язок між моделями життєвого циклу та підходами до управління є критичним для успіху програмного проєкту. Правильний вибір і адаптація моделі життєвого циклу під специфіку проєкту і відповідний підхід до управління допомагають досягти високих показників ефективності, якості і задоволення замовника. Розуміння цих взаємозв'язків дозволяє правильно планувати ресурси, уникати помилок на етапах розробки і досягати успіху навіть у складних і змінних умовах.

1.1.3 Неструктурований життєвий цикл програмних проєктів

На початкових етапах розвитку сфери розробки програмного забезпечення сам процес створення програм не мав чіткої структури чи визначених етапів. Він будувався за досить простою схемою: спочатку формувалось завдання, а потім розпочиналася його реалізація, яка тривала доти, доки завдання не було виконане або не втрачало актуальність і, відповідно, скасовувалося.

Такий підхід вважався прийнятним і ефективним у ті часи, коли програмне забезпечення створювалося переважно для вузькоспеціалізованих цілей, зокрема для проведення наукових розрахунків та обробки математичних даних. Подібні програми, як правило, не відзначалися великою складністю своєї структури, мали порівняно невеликий обсяг коду та розроблялися для вирішення конкретних, добре визначених завдань.

Проте, зі збільшенням масштабів програмних проєктів, ускладненням їхньої структури та зростанням вимог до якості та надійності програмного забезпечення стало очевидним, що неструктурована модель життєвого циклу більше не може задовольняти потреби розробників і замовників. Вона не враховувала низку критично важливих процесів, необхідних для ефективного створення, тестування та підтримки великих і комплексних програмних продуктів. Як наслідок, постала необхідність у розробці більш впорядкованих, системних підходів до організації процесу розробки програмного забезпечення.

1.1.4 Каскадна модель життєвого циклу програмних проєктів

Як було зазначено раніше, перша загальновизнана модель життєвого циклу програмного проєкту - це модель SLC (Software Life Cycle). Ця модель поклала початок багатьом підходам до організації та управління процесом розробки програмного забезпечення, пропонуючи чітку послідовність етапів. Однак, незважаючи на те, що етапи цієї моделі традиційно не передбачають повернення на попередні стадії, на практиці багато компаній-розробників програмного забезпечення все ж здійснюють певні коригування і адаптації в процесі, що дозволяє їм повернутися до попередніх етапів для вдосконалення чи виправлення недоліків [2].

Попри те, що модель SLC на сьогодні вважається застарілою і практично не застосовується в своєму початковому вигляді, вона стала основою для розробки багатьох сучасних моделей життєвого циклу розробки ПЗ. Ключові принципи, термінологія та основні етапи цієї моделі були успішно адаптовані та інтегровані в більш сучасні та вдосконалені підходи, що дозволяють врахувати сучасні вимоги до гнучкості та адаптивності процесу розробки. Таким чином, хоча сама модель SLC більше не використовується в її первісному вигляді, її основи залишаються актуальними і надалі відіграють важливу роль у формуванні сучасних підходів до оцінки та управління проєктами з розробки програмного забезпечення.

1.1.5 V-подібна модель життєвого циклу

На зміну традиційній каскадній моделі було запропоновано більш сучасний підхід - V-подібну (шарнірну) модель, основною метою якої було усунення значних недоліків каскадної моделі. Зокрема, нова модель мала на

меті більш ефективно поєднати етапи постановки завдання з перевіркою його відповідності специфікаціям, що дозволяло знизити ризики на етапах розробки [3].

Проте, незважаючи на ці вдосконалення, «інженерний» підхід, що був закладений в обох моделях - як SLC, так і V-подібній, - виявився недостатньо гнучким і ефективним для вирішення потреб великих проєктів. Тривала поетапна розробка, що є однією з характеристик цих моделей, часто призводила до серйозних перевищень як бюджету, так і часу, визначених на самому початку проєкту. Крім того, кінцевий продукт нерідко не встигав вийти на ринок у запланований термін, що ставало серйозною проблемою для компаній, що прагнули залишатися конкурентоспроможними в умовах швидко змінюваного ринку.

З точки зору інноваційного менеджменту, варто зазначити, що моделі SLC та V-подібна модель мають суттєві обмеження, коли йдеться про впровадження ефективного управління інноваціями. Реалізація проєкту за цими моделями може тривати багато років, і протягом цього часу науково-технічний прогрес не враховується в достатній мірі. Вимоги та плани реалізації, які розробляються на самому початку проєкту, ґрунтуються на технологіях і підходах, актуальних на момент старту робіт, але, як показує практика, вони не враховують стрімкий розвиток технологій, що відбувається протягом всього процесу розробки. Як наслідок, кінцевий продукт часто виявляється морально застарілим ще до того, як він потрапляє на ринок, і не здатний конкурувати з більш сучасними рішеннями.

1.1.6 Модель життєвого циклу на основі розробки прототипів

Модель життєвого циклу, яка ґрунтується на розробці прототипів, передбачає створення як повністю функціонуючих, так і частково

реалізованих моделей майбутньої системи. Такий підхід дозволяє на ранніх етапах візуалізувати й оцінити основні аспекти майбутнього продукту, надаючи команді можливість коригувати проєкт ще до того, як розпочнеться повномасштабна розробка. Це нагадує використання зменшених моделей будівель в архітектурі, де на основі макетів можна виявити недоліки та внести корективи на ранньому етапі проєктування.

Однак основним недоліком цієї моделі є ризик виникнення конфлікту інтересів. Зокрема, існує ймовірність того, що через постійне вдосконалення прототипу команда може потрапити в нескінченний цикл коригувань, що ускладнює процес прийняття фінальних рішень. Це може призвести до значних затримок у розробці та, зрештою, до відстрочки виходу готового продукту на ринок.

1.1.7 Інкрементна модель життєвого циклу програмного проєкту

Інкрементна модель життєвого циклу є підходом, що передбачає поступову розробку продукту шляхом постачання його частинами, де кожен новий етап додає до функціональності попереднього. Такий підхід дозволяє на кожному кроці досягати певних результатів, які можуть бути представлені замовнику для оцінки та зворотного зв'язку [4].

Однією з головних переваг інкрементної моделі є те, що вона дає змогу поступово просуватися вперед, забезпечуючи при цьому баланс між вимогами замовника та можливостями команди розробників. Завдяки поетапній реалізації проєкту, усі сторони можуть оцінити результати на кожному етапі і, при необхідності, коригувати напрямок роботи без необхідності чекати завершення повної розробки.

З іншого боку, інкрементна модель має свої недоліки. Оскільки проєкт розвивається етапами, з кожним новим інкрементом зростає складність і

потреба у більш точному плануванні, що може призвести до збільшення вартості та часу, необхідних для повної реалізації продукту. Це означає, що з кожним новим етапом проєкт може ставати все більш складним у реалізації, що в кінцевому підсумку може вплинути на загальні ресурси, потрібні для завершення розробки.

1.1.8 Спіральна модель проєктування

Спіральна модель життєвого циклу була розроблена для того, щоб подолати основні недоліки традиційної моделі SLC, яка виявилася недостатньо гнучкою для вирішення змінюваних умов проєктів, що з’являються в процесі розробки. Згідно з Б. Боемом, спіральна модель є надзвичайно гнучким та адаптивним підходом до управління проєктом, де кожен етап дозволяє враховувати можливі ризики та вчасно реагувати на їх появу, що значно зменшує ймовірність виникнення серйозних проблем у процесі розробки. Ця модель здобула популярність особливо в великих проєктах з розробки програмного забезпечення, де кілька груп розробників працюють над різними аспектами проєкту.

Основними перевагами спіральної моделі є її циклічний підхід, що забезпечує поступове і безпечне збільшення рівня визначеності щодо функціональності продукту, а також здатність знижувати рівень ризиків на кожному етапі проєкту. Такий підхід дозволяє адаптуватися до непередбачених ситуацій та мінімізувати негативні наслідки, пов’язані з недооцінкою або ігноруванням можливих проблем на ранніх етапах. Іншою важливою особливістю є наявність чітко визначених контрольних точок або віх, які дозволяють не тільки відслідковувати хід проєкту, але й погоджувати рішення з усіма сторонами проєкту, що важливо для досягнення згоди та підтримки комунікації.

У спіральній моделі ризики розглядаються як потенційні перешкоди на шляху досягнення визначених цілей проєкту. Кожен етап оцінки ризиків є не лише запобіжним механізмом, але й способом підвищення гнучкості проєкту, адже на цьому етапі розглядаються можливі шляхи вирішення проблем, які можуть виникнути в майбутньому. Так, на кожній стадії модель дозволяє оцінювати нові ризики, що дає змогу не тільки знижувати їх вплив на кінцевий результат, але й вчасно реагувати на зміни зовнішніх і внутрішніх факторів.

Спіральний підхід виявляється особливо ефективним для управління інноваціями, оскільки кожен виток спіралі починається з етапу оцінки ризиків, що дозволяє виявити нові технології чи інновації, які можуть значно підвищити ефективність проєкту. Прогнозування та оцінка таких можливостей дає можливість вибрати найбільш перспективні та економічно вигідні шляхи розвитку продукту.

Цей підхід до управління проєктом можна проілюструвати прикладом, який стався на початку 1980-х років. Одна компанія-замовник розробляла інформаційну систему для більше ніж тисячі користувачів, і вимоги до швидкості її реакції були надзвичайно високими. За допомогою класичної каскадної моделі проєктування було виявлено, що для досягнення встановлених вимог потрібно величезних витрат. Однак, використовуючи спіральну модель, фахівці змогли змінити вимоги до швидкості реакції та значно знизити витрати на проєкт, а також знайти більш оптимальну архітектуру системи. Це дозволило зменшити вартість розробки проєкту майже втричі.

Замість того, щоб сліпо слідувати за початковими вимогами, спіральна модель дає змогу кожен етап проєктування переглядати і коригувати з урахуванням нових ризиків і можливостей, що виникають у процесі роботи. Це дозволяє не тільки знизити фінансові витрати, але й забезпечити високий рівень конкурентоспроможності кінцевого продукту на ринку, оскільки з кожним витком спіралі модель дозволяє удосконалювати продукт та адаптувати його до змінюваних умов [5].

1.1.9 Ітеративна модель життєвого циклу програмного проєкту

Подальший розвиток ідей спіральної моделі розпочався з впровадження так званого «ітеративного підходу», який вперше був запропонований Ф. Крачтеном. Оригінальна спіральна модель, запропонована Б. Боемом, використовує ітеративний підхід лише для процесу проєктування, в той час як інші етапи розробки програмного забезпечення залишаються послідовними, як у класичній каскадній моделі. Це означає, що окремі етапи розробки виконуються один за одним, без значної корекції на попередніх етапах. Ф. Крачтен, у свою чергу, значно вдосконалив цю концепцію, поширивши ітеративний підхід на всі етапи розробки програмного забезпечення, що призвело до виділення чотирьох основних фаз життєвого циклу [6].

а) Початок. На цьому етапі команда проєкту концентрується на розумінні бізнес-варіанту проєкту, визначенні його масштабу та потенціалу для реалізації. Проводиться первинний аналіз проблеми, розробляється концептуальний документ, який включає в себе попередні оцінки термінів, бюджету та можливих ризиків. Ця фаза передбачає ретельне планування та оцінку проєкту для того, щоб визначити, чи варто рухатися вперед.

б) Дослідження. На цьому етапі уточнюються вимоги до майбутньої системи, визначається її базова архітектура, а також створюється попередній прототип. Це важливий етап, на якому тестуються перші ідеї і формується основа для подальшої розробки.

в) Побудова. Основна увага на цьому етапі зосереджена на реалізації. Це етап, на якому створюється основний код, завершується проєктування та доопрацьовується архітектура системи. Важливо, щоб усі компоненти працювали коректно та відповідали вимогам, що були визначені на попередніх етапах.

г) Впровадження. Після того як система готова, проводиться бета-тестування, на якому користувачі та команда супроводу отримують досвід

роботи з системою. Це дозволяє виявити та виправити можливі недоліки, перш ніж система стане доступною для широкого використання. Протестована та вдосконалена версія ПЗ передається кінцевим користувачам і розгортається для їхнього використання.

Ітеративна модель передбачає, що після кожної ітерації виводиться нова життєздатна версія системи, що дозволяє систематично і безперервно вдосконалювати продукт протягом усіх етапів його розробки. Ітерації охоплюють не лише процес проєктування, а й всі інші важливі етапи, де це можливо. Процеси в рамках ітеративної моделі можуть бути розділені на основні та допоміжні, причому кожен з них може мати власний цикл ітерацій, що не завжди збігається з циклами інших процесів.

У зв’язку з тим, що ітеративна модель значно розширила та удосконалила основні положення спіральної моделі, вона фактично замінила оригінальну модель, запропоновану Б. Боемом. Відтепер у сучасних умовах під терміном «спіральна модель» зазвичай мається на увазі не оригінальна модель Б. Боема, а саме ітеративна модель Ф. Крачтена. Цей підхід набув широкого застосування не лише в розробці програмного забезпечення, а й у загальному управлінні проєктами. Він став стандартом у багатьох сферах, де необхідно ефективно управляти складними проєктами.

У 1998 році Б. Боем разом з іншими дослідниками запропонував модифікацію спіральної моделі, яку було названо спіральною моделлю «Win-Win». Ця модель фактично є відображенням гри між двома учасниками, причому умовою «виграшу» є успішне завершення проєкту, що приносить користь усім сторонам. У такому підході досягнення мети є спільною вигодою, що визначає успіх проєкту.

На нашу думку, як спіральна, так і ітеративна моделі мають значний потенціал для ефективного впровадження інновацій в процес розробки програмного забезпечення. Кожна нова ітерація може стати чудовою можливістю для впровадження нових ідей та рішень, що можуть значно підвищити ефективність проєкту. Однак для цього важливо розпочати кожну

ітерацію з процесу вибору та оцінки нововведень, що дозволяють досягти максимального економіко-технічного ефекту як для конкретного проєкту, так і для фірми-розробника загалом. Врахування можливих ризиків, таких як затримки в реалізації проєкту через впровадження інновацій, дозволяє знизити ймовірність неочікуваних негативних наслідків і забезпечити стабільний розвиток проєкту.

1.2 Гнучкі методології управління проєктом

Гнучкі методології управління проєктами (Agile methodologies) ґрунтуються на концепції ітеративної розробки, постійному вдосконаленні процесів і безперервній взаємодії між замовником і командою. В основі цих методологій лежить прагнення до максимального задоволення потреб клієнтів шляхом гнучкості в плануванні, що дозволяє швидко реагувати на зміни як у вимогах до продукту, так і на зовнішні фактори.

Одним із центральних елементів гнучких методологій є ітераційний процес, що передбачає регулярне створення робочих версій продукту, які можна тестувати, коригувати та удосконалювати. Кожна ітерація зазвичай триває кілька тижнів (наприклад, 2-4 тижні), і на її кінець команда має доставити клієнту функціональний результат.

Основні принципи гнучких методологій

Згідно з Маніфестом Agile, який був опублікований у 2001 році, гнучкі методології визначають чотири основні цінності та дванадцять принципів. Основні цінності:

- індивідуальність та взаємодія важливіші за процеси та інструменти;
- працюючий продукт важливіший за детальну документацію;
- співпраця з клієнтом важливіша за контрактні умови;
- готовність до змін важливіша за слідування плану.

Ці цінності формулюють основний підхід до управління проєктами, який дозволяє зосередити зусилля на ефективній взаємодії між членами команди та замовником, створенні продукту, який працює, а також гнучкості при необхідності змінювати напрямок роботи [7].

1.2.1 Scrum

Scrum - це методологія управління проєктами, яка широко використовується в розробці програмного забезпечення. Вона дозволяє командам швидко адаптуватися до змін і постійно покращувати продукт через регулярний зворотний зв'язок від замовника і користувачів. В основі Scrum лежить принцип ітераційного розвитку, де проєкт розвивається поступово, кожен спринт додає нові функції та вдосконалення [8].

У Scrum є три ключові ролі. Scrum Master допомагає команді працювати за методологією Scrum, усуваючи перешкоди та забезпечуючи ефективну роботу. Product Owner відповідає за визначення пріоритетів завдань та забезпечує зв'язок між командою і замовником, створюючи список необхідних функцій для продукту - Product Backlog. Команда розробників працює безпосередньо над виконанням завдань, визначених у спринті.

Процес Scrum організований у цикли, які називаються спринтами. Кожен спринт триває від двох до чотирьох тижнів, і на його початку команда визначає, які завдання будуть виконані, створюючи список завдань для цього спринту - Sprint Backlog. Щодня команда проводить коротку зустріч, де обговорюється прогрес і перешкоди. В кінці спринту відбувається огляд результатів роботи та ретроспектива, щоб оцінити, що вдалося, а що можна покращити в наступних спринтах.

Scrum використовує два основних підходи: ітеративний, коли продукт вдосконалюється на кожному етапі, та інкрементальний, коли продукт

розвивається по частинах, додаючи нові можливості з кожним спринтом. Це дозволяє команді бути гнучкою та швидко реагувати на зміни в вимогах або обставинах.

Основні артефакти Scrum - це Product Backlog, який містить усі завдання для продукту, Sprint Backlog, який містить завдання на поточний спринт, та Increment, що відображає частину готового продукту після кожного спринту.

Основними перевагами Scrum є гнучкість, що дозволяє швидко адаптуватися до змін, а також можливість регулярно перевіряти і коригувати продукт, що підвищує його якість. Крім того, Scrum сприяє відкритій комунікації, самостійній організації роботи команди та забезпечує прозорість на кожному етапі розробки.

1.2.2 Kanban

Kanban - це методологія управління проєктами, яка зосереджена на візуалізації робочих процесів і постійному вдосконаленні. Вона використовується для покращення ефективності та зменшення часу виконання завдань у різних сферах діяльності, особливо в розробці програмного забезпечення. Канбан дозволяє команді оптимізувати робочі процеси, зменшити витрати часу на управління і підвищити продуктивність через постійну адаптацію та зменшення перевантаження [9].

Основною ідеєю Kanban є візуалізація всіх завдань на спеціальній дошці, де кожне завдання або етап роботи представлений окремою карткою. Канбан-дошка розділена на кілька колонок, що відображають різні стадії робочого процесу, наприклад, "To Do", "In Progress", "Done". Така візуалізація допомагає команді швидко бачити поточний стан завдань, виявляти вузькі місця і ухвалювати рішення для покращення продуктивності.

Одним з основних принципів Kanban є обмеження кількості завдань, які

можуть перебувати на кожному етапі роботи. Це допомагає уникнути перевантаження команди і забезпечує постійний потік завдань. Наприклад, на етапі "In Progress" може бути обмежена кількість одночасних завдань, щоб команда могла зосередитися на їх завершенні, а не брати на себе нові завдання, не закінчивши попередні.

Kanban не передбачає чітких ітерацій, як в Scrum. Натомість процес роботи в Kanban є безперервним: як тільки завдання завершується, нове може бути розпочате. Це дозволяє команді більш гнучко реагувати на зміни і не прив'язуватися до фіксованих періодів, як у спринтах.

Крім того, Kanban заохочує до постійного покращення процесів. Команди регулярно аналізують свої робочі потоки, шукають способи зменшити час виконання завдань, підвищити ефективність і зменшити відставання.

Основними перевагами Kanban є гнучкість і простота впровадження. Ця методологія підходить для команд, де робочі процеси не можна чітко розбити на етапи спринтів, або де завдання мають змінюватися дуже часто. Канбан дозволяє підтримувати високу прозорість роботи і легко адаптуватися до нових вимог чи змін у пріоритетах.

У результаті Kanban дозволяє зменшити час циклу, зменшити витрати часу на планування і покращити загальну ефективність роботи команди.

1.2.3 Extreme Programming

Extreme Programming - це методологія гнучкої розробки програмного забезпечення, яка зосереджена на технічних аспектах розробки і високій якості коду. XP ставить на перше місце ефективну комунікацію в команді, швидку адаптацію до змін та максимальну взаємодію з користувачами. Цей підхід допомагає забезпечити високу якість програмного забезпечення, постійно

тестуючи код і виконуючи необхідні зміни на ранніх етапах розробки [10].

Один з основних принципів XP - це розробка через тестування (Test-Driven Development, TDD). У цьому підході програмісти спочатку пишуть тести для нових функцій, а потім створюють код, який ці тести проходить. Такий підхід забезпечує високий рівень тестування коду, допомагаючи уникнути помилок і дефектів на ранніх етапах.

Ще однією важливою характеристикою XP є парне програмування (Pair Programming). Це метод, при якому два розробники працюють за одним комп’ютером: один пише код, а інший переглядає його, даючи зворотний зв'язок та пропонуючи покращення. Це не тільки дозволяє підвищити якість коду, але й сприяє кращому навчанні і взаємодії між розробниками.

XP також зосереджений на частих релізах продукту, щоб клієнти могли отримати працюючу версію програмного забезпечення через короткі періоди часу (від одного до кількох тижнів). Це дає змогу отримувати зворотний зв'язок від користувачів та оперативно вносити корективи в продукт.

Крім того, XP передбачає простий дизайн і рефакторинг. Розробники повинні створювати простий, чистий код, який легко змінювати в майбутньому. Рефакторинг, або постійне вдосконалення коду, дозволяє підтримувати високу якість програмного забезпечення без значних витрат на переписування.

Ще одним важливим аспектом XP є безперервне інтегрування. У цьому підході програмісти часто інтегрують свої зміни в загальний код, що дозволяє уникнути проблем із сумісністю та знижує ймовірність виникнення конфліктів між різними частинами програмного забезпечення.

Основними перевагами Extreme Programming є високий рівень якості програмного забезпечення, здатність швидко реагувати на зміни вимог і підвищення ефективності команди завдяки тісній взаємодії між її учасниками. Такий підхід особливо підходить для проєктів, де вимоги часто змінюються, і де необхідно забезпечити високу гнучкість і адаптивність.

Extreme Programming дозволяє командам не тільки ефективно розвивати

програмне забезпечення, а й створювати сильну командну культуру, зосереджену на співпраці, тестуванні та покращенні якості коду на всіх етапах розробки.

1.2.4 Lean

Lean - це методологія управління проєктами, яка виникла з виробничого процесу в Японії, зокрема, в компанії Toyota, і з часом адаптувалася для застосування в різних галузях, зокрема в розробці програмного забезпечення. Основна мета Lean - максимізувати цінність для клієнта, одночасно знижуючи витрати і втрачені можливості. Цей підхід зосереджується на оптимізації процесів, усуненні непотрібних витрат і покращенні якості на кожному етапі розробки [11].

Один з основних принципів Lean - це мінімізація втрат. Під "втратами" розуміються будь-які ресурси, що витрачаються не на створення цінності для клієнта. Втрати можуть включати в себе зайву роботу, надмірні запаси, час на очікування, надмірні зміни в процесах тощо. Lean сприяє виявленню і усуненню цих втрат для покращення ефективності.

Lean також передбачає безперервне вдосконалення процесів - Kaizen. Це концепція, яка вимагає постійного вдосконалення на всіх рівнях організації, будь то продукт, процес або робоче середовище. Завдяки цьому підходу команда постійно шукає нові способи підвищення ефективності та зниження витрат.

Іншим важливим аспектом Lean є ефективність потоку роботи. Це означає, що всі етапи робочого процесу повинні бути оптимізовані так, щоб завдання рухалися через них без затримок, що може спричинити непотрібне накопичення роботи і зниження продуктивності. Lean використовує концепцію Pull system, коли робота розпочинається тільки після того, як на це

є запит або потреба, а не на основі запланованих термінів.

Lean також сприяє спрощенню процесів. Всі надлишкові етапи та складні процедури усуваються, що дозволяє знизити час, витрачений на виконання завдань. Крім того, команда має зосереджуватися на найважливіших завданнях, які дають найбільшу цінність для користувача.

Ще одним принципом Lean є залучення всіх членів команди до процесу покращення. Всі співробітники мають брати участь у пошуку можливостей для вдосконалення, що допомагає створювати культуру співпраці та постійного навчання.

Основними перевагами Lean є високий рівень ефективності, зниження витрат і часу на виконання завдань, а також покращення якості продукту. Оскільки Lean спрямований на оптимізацію всіх етапів процесу, він дозволяє знижувати ризики і забезпечувати стійкий розвиток команди та продукту.

Lean підходить для проєктів, де важливо максимально ефективно використовувати ресурси, швидко реагувати на зміни та постійно покращувати процеси. Завдяки гнучкості та можливості швидко адаптуватися до змін, методологія Lean допомагає забезпечити високий рівень задоволеності клієнтів і досягнення бізнес-цілей.

1.2.5 Scrumban

Scrumban - це сучасна методологія управління проєктами, яка поєднує в собі елементи двох популярних підходів: Scrum і Kanban. Вона виникла як практичне рішення для команд, яким необхідна більша гнучкість, ніж надає класичний Scrum, але які все ще хочуть зберегти певну структуру в організації процесу. У той час як Scrum базується на роботі в чітко визначених ітераціях - спринтах, Scrumban відмовляється від жорсткої циклічності на користь більш гнучкого, адаптивного підходу до планування. У цьому підході завдання не

розподіляються строго по спринтах, а виконуються у міру звільнення ресурсів, що дозволяє уникнути простоїв та краще реагувати на нові вимоги [12].

В основі Scrumban лежить принцип візуалізації робочого процесу. Для цього команда використовує канбан-дошку, яка поділена на колонки відповідно до етапів виконання роботи, наприклад: «Заплановано», «У роботі», «Готово». Завдання переміщуються по дошці в міру просування, що дає можливість команді й замовнику у будь-який момент бачити реальний стан справ. Однією з ключових характеристик Scrumban є обмеження кількості задач, які можуть одночасно перебувати в роботі. Це допомагає команді зосередитися на завершенні поточних завдань, а не на постійному старті нових, що підвищує якість результатів і зменшує кількість незавершених справ.

Ще однією важливою особливістю є підхід до планування. У Scrumban немає фіксованого циклу планувань, як у Scrum. Натомість планування відбувається на вимогу - коли кількість завдань у колонці «Заплановано» падає нижче певного мінімального рівня, команда проводить коротку планувальну сесію, щоб додати нові задачі. Це дозволяє більш гнучко управляти пріоритетами, адаптуючись до змін в реальному часі. При цьому, якщо команда вважає корисними деякі елементи Scrum, наприклад, щоденні стендапи, ретроспективи чи роль Product Owner, вона може вільно впроваджувати їх у свою практику. Таким чином, методологія Scrumban не є жорстко регламентованою - вона адаптується під потреби конкретної команди, дозволяючи зберігати ефективність навіть у складних і динамічних проєктах.

Scrumban особливо добре підходить для команд, які працюють із постійно змінними запитами, обслуговують підтримку або займаються довготривалими проєктами без чітко визначених етапів. Він забезпечує баланс між структурованістю та свободою дій, дозволяючи командам одночасно планувати, виконувати та вдосконалювати свою роботу без зайвого навантаження. У результаті, цей підхід сприяє підвищенню продуктивності, прозорості й командної відповідальності.

1.2.6 Crystal

Crystal - це гнучка (agile) методологія управління проєктами, яка фокусується на адаптації процесу під конкретну команду, її розмір, критичність проєкту та характер взаємодії. Її автор - Алістер Коберн (Alistair Cockburn), один із підписантів Маніфесту Agile. Crystal вирізняється тим, що не має єдиного набору жорстких правил чи ролей - натомість вона пропонує набір принципів і практик, які можна варіювати залежно від контексту. Метою Crystal є мінімізація бюрократії та максимізація ефективності шляхом підлаштування процесу під людей, а не навпаки [13].

Методологія Crystal базується на припущенні, що кожен проєкт є унікальним і не може бути ефективно реалізований за універсальним підходом. Тому Crystal розроблена як сімейство методологій, які позначаються кольорами відповідно до складності й розміру проєкту. Наприклад, Crystal Clear підходить для невеликих команд (до 6-8 осіб), які працюють над некритичними проєктами, тоді як Crystal Orange або Crystal Red орієнтовані на більші проєкти з вищими вимогами до надійності. Чим "темніший" колір у системі Crystal, тим більше формальних практик і дисципліни потрібно застосовувати.

Ключовими принципами Crystal є ефективна комунікація в команді, самоперевірка і рефлексія, регулярні інтерв’ю або зустрічі для обговорення прогресу, а також швидка реакція на зміни. Команда має свободу у виборі інструментів і технік, які найкраще відповідають її стилю роботи. У Crystal важливою є прозорість роботи та особиста відповідальність кожного учасника. Більшість практик спрямовані на підтримку інформаційного обміну, наприклад, використання інформативних просторів (інформаційні стіни), часті перегляди коду або демонстрації проміжного результату.

Crystal не заперечує використання артефактів, таких як документація, діаграми чи звіти, але акцент робиться лише на тих, які дійсно приносять

цінність. Тому однією з головних переваг методології є її легкість і адаптивність - вона дозволяє команді уникати зайвих процесів і зосередитися на розробці якісного продукту. Crystal добре підходить для команд, які вже мають певний рівень досвіду та автономії, а також для середовищ, де важлива швидкість реакції та пряма взаємодія між учасниками розробки.

Ця методологія отримала визнання серед тих організацій, які прагнуть уникнути надмірної формалізації процесів і водночас підтримувати високу якість продукту та взаємодії в команді. Crystal не нав’язує суворих ролей або церемоній, дозволяючи створювати індивідуальний робочий процес, який найкраще відповідає реаліям проєкту.

1.3 Аналіз наявних інформаційних систем і технологій гнучкого управління IT проєктами

У сучасних умовах керівники організацій постійно стикаються з різними управлінськими викликами, які впливають на ефективність роботи підприємства та його здатність адаптуватися до змін. Зокрема, йдеться про необхідність грамотного планування робочого часу співробітників, раціонального розподілу ресурсів у потрібному обсязі та в установлені терміни, а також контролю витрат і забезпечення оптимального використання наявних можливостей.

Вирішення цих питань є складним завданням, яке вимагає системного підходу та застосування сучасних методів управління. Одним із найбільш ефективних підходів є проєктне управління, яке дозволяє оптимізувати процеси, підвищити продуктивність та забезпечити якісне виконання поставлених завдань. Використання такого підходу сприяє покращенню координації між співробітниками, більш чіткому плануванню діяльності та забезпеченню стабільного розвитку організації. Саме тому проєктне

управління сьогодні набуває все більшого значення і є важливою складовою успішного функціонування сучасних компаній.

Окремо варто відзначити, що системи управління проєктами дедалі частіше впроваджуються у діяльність підприємств, оскільки вони відіграють ключову роль у досягненні поставлених цілей та забезпечують ефективність усіх процесів. Завдяки використанню таких систем керівники отримують можливість не лише контролювати виконання завдань, а й оперативно реагувати на зміни, приймати виважені рішення та забезпечувати високий рівень організованості в роботі.

Однак для того, щоб управління проєктами було максимально ефективним, важливо застосовувати сучасні інформаційні технології, зокрема онлайн-платформи та спеціалізовані сервіси. Без них у сучасних умовах практично неможливо досягти високого рівня економічності, доступності інформації, а також забезпечити відображення всіх процесів у режимі реального часу. Використання таких інструментів дозволяє значно покращити координацію між учасниками проєктів, мінімізувати ризики, пов’язані з невчасним виконанням завдань, та підвищити загальну продуктивність організації.

На основі аналізу ринку онлайн-сервісів для управління проєктами доцільно виокремити ключові платформи, які допоможуть знайти оптимальне рішення для застосування інформаційних систем у цій сфері. Серед найпопулярніших інструментів для управління проєктами можна виділити п’ять платформ: Asana, Trello, Todoist, Jira та Teamwork.

1.3.1 Asana

Asana - гнучкий інструмент для управління завданнями та проєктами [14].

Сервіс Asana, створений у 2008 році, призначений для командної роботи над завданнями, забезпечуючи ефективну комунікацію між співробітниками, можливість обміну файлами та організацію робочого процесу. Інтерфейс платформи дозволяє відображати завдання в різних форматах: список, kanban-дошка, календар, таймлайн, дашборд, Workflow (рисунок 1.1) (таблиця 1.1).

Основні функції Asana:

− створення та управління завданнями дозволяє чітко визначати обсяг робіт та відповідальних осіб;

− призначення відповідальних осіб та встановлення пріоритетів сприяє ефективному розподілу обов'язків та фокусуванню на важливих завданнях;

− надсилання повідомлень та запрошення учасників у проєкт забезпечує безперебійну комунікацію та залученість команди;

− інтеграція з популярними сервісами такими як Gmail, Zoom, Slack тощо, що дозволяє об'єднати різні інструменти в єдину екосистему;

− базові функції доступні безкоштовно, розширені можливості - у платних тарифах.

Таблиця 1.1 - Порівняння цінових планів Asana

Тарифний план	Вартість	Основні можливості
1	2	3
Personal (безкоштовний)	$0	До 10 користувачів, базові функції управління проєктами, списки завдань, обмін файлами, базові інтеграції.

Кінець таблиці 1.1

1	2	3
Starter	$10.99/міс (річна оплата) $13.49/міс (щомісячна оплата)	Кастомні поля, розширені звіти, таймлайн (діаграма Ганта), форми, автоматизація процесів.
Advanced	$24.99/міс (річна оплата) $30.49/міс (щомісячна оплата)	Всі можливості Starter + портфелі проєктів, цілі, робочі навантаження, розширені інтеграції, підтримка пріоритетних клієнтів.
Enterprise	Індивідуальна ціна	Максимальні можливості: підвищена безпека, корпоративні інтеграції, спеціальні налаштування та персоналізована підтримка.

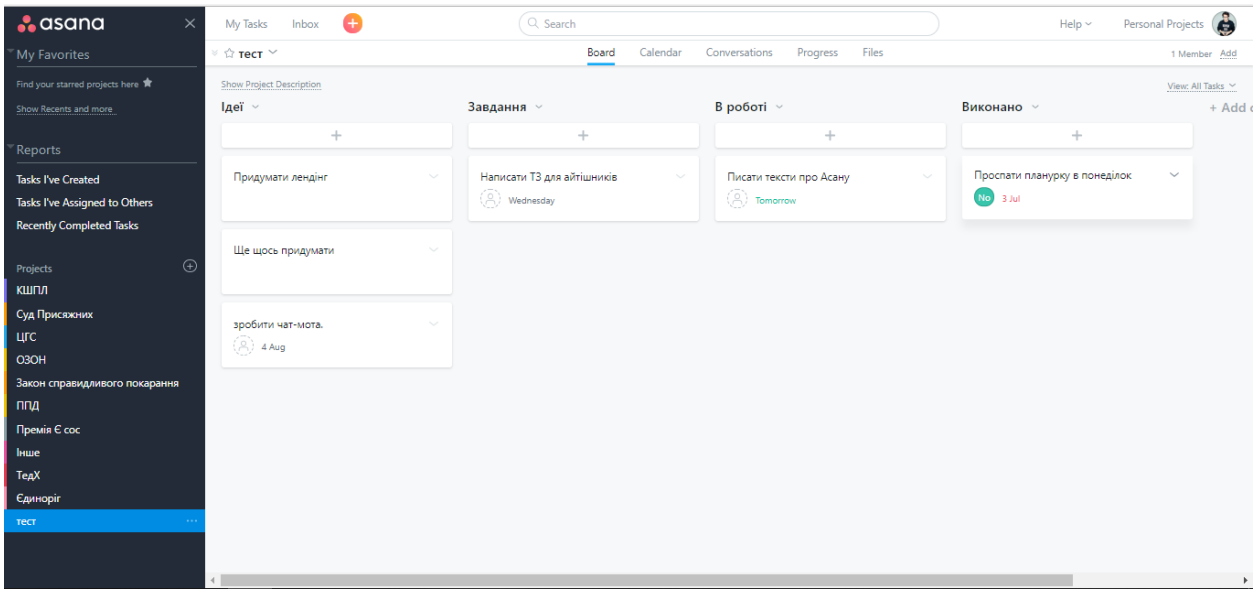

Рисунок 1.1 - Приклад застосунку Asana

1.3.2 Trello

Trello - це популярний сервіс для управління завданнями, створений у 2011 році. Він доступний як у веб-версії, так і у вигляді мобільного додатку для смартфонів. Основою роботи сервісу є Kanban-дошки, які забезпечують візуалізацію завдань та їх статусів у процесі виконання (рисунок 1.2) [15].

Основні складові робочого простору Trello:

− дошка - головний робочий екран, на якому відображаються всі завдання та їх статуси;

− список - стовпець, у якому згруповані завдання певного етапу чи категорії;

− картка - окреме завдання, що містить опис, чек-листи, прикріплені файли, дедлайни та виконавців.

Користувачі можуть створювати необмежену кількість карток і списків у межах однієї дошки, а також керувати видимістю дощок (приватна, командна або публічна) (таблиця 1.2).

Таблиця 1.2 - Порівняння цінових планів Trello

Тарифний план	Вартість (при річній оплаті)	Основні можливості
1	2	3
Free	Безкоштовно	До 10 дощок на робочий простір, необмежена кількість карток, необмежені Power-Ups на дошку, обмеження на розмір вкладених файлів до 10 МБ, 250 автоматизацій команд на місяць.

Кінець таблиці 1.2

1	2	3
Standard	$5 за користувача на місяць	Усі можливості Free, плюс: необмежена кількість дощок, розширені чек-листи, кастомні поля, збільшений розмір вкладених файлів до 250 МБ, 1 000 автоматизацій команд на місяць.
Premium	$10 за користувача на місяць	Усі можливості Standard, плюс: додаткові види відображення (Календар, Хронологія, Таблиця, Панель, Карта), необмежена кількість автоматизацій, розширені функції адміністрування та безпеки, експорт даних.
Enterprise	$17.50 за користувача на місяць	Усі можливості Premium, плюс: необмежена кількість робочих просторів, організаційні дозволи, управління публічними дошками, багатодошкові гості, розширені налаштування безпеки та адміністрування.

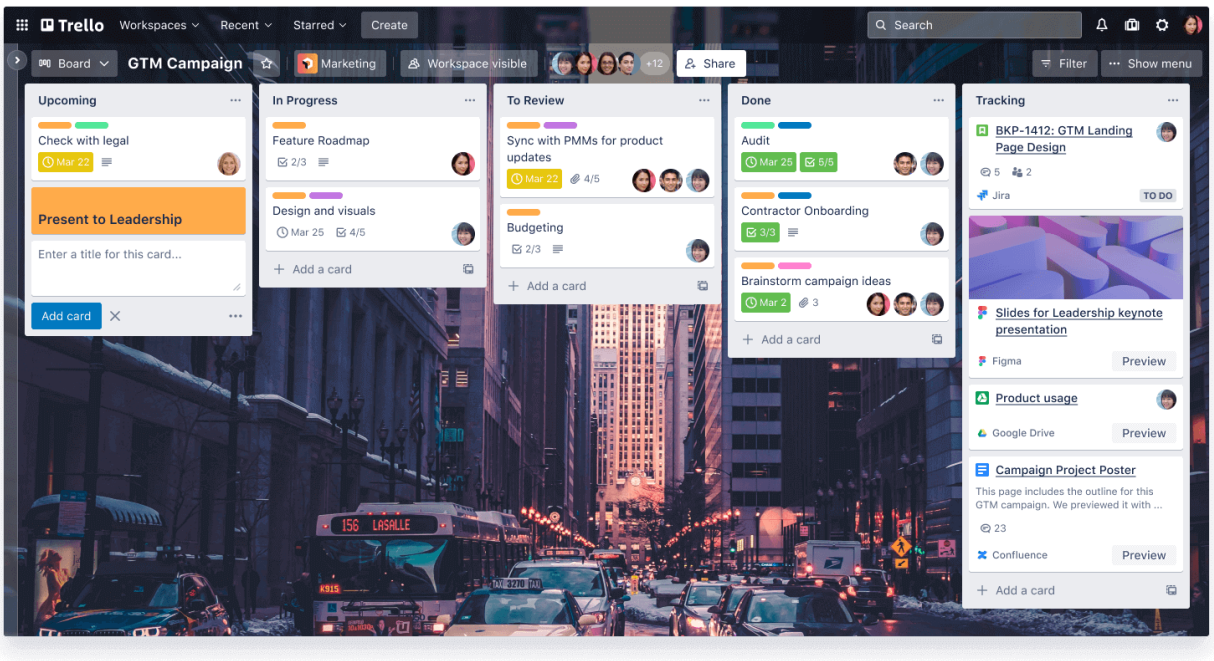

Рисунок 1.2 - Приклад застосунку Trello

1.3.3 Todoist

Todoist - це потужний менеджер завдань, який можна використовувати як для особистого планування, так і для командної роботи. Він був створений у 2007 році компанією Doist і з того часу здобув велику популярність завдяки своїй простоті та функціональності (рисунок 1.3) [16].

Todoist дозволяє організовувати завдання, призначати відповідальних і встановлювати терміни виконання. Користувачі можуть створювати підзадачі, розподіляти завдання по проєктах, а також позначати їх пріоритетність. Важливою особливістю є можливість роботи з повторюваними завданнями, що корисно для планування регулярних справ. Завдання можна супроводжувати коментарями, обмінюватися файлами та отримувати сповіщення про їх виконання.

Для команд Todoist пропонує можливість централізованого управління проєктами, де можна призначати учасників для кожного завдання, обговорювати деталі прямо в додатку і навіть інтегрувати інші сервіси, такі як

Google Calendar або Slack. Це дозволяє ефективно координувати роботу команди та зберігати всю важливу інформацію в одному місці.

Todoist також дає змогу бачити прогрес за допомогою аналітичних звітів і графіків, що допомагає відстежувати ефективність виконання завдань і мотивує користувачів до більшої продуктивності. Крім того, програма доступна на всіх популярних платформах: веб-версія, додатки для Windows, macOS, iOS, Android, а також для пристроїв Apple Watch і Wear OS.

Todoist має різні тарифні плани, серед яких є безкоштовний варіант з обмеженими функціями, а також платні тарифи, які пропонують більш розширені можливості, такі як більша кількість проєктів, можливість додавати більше співпрацівників і зберігати більше файлів. Платний план Business також включає додаткові функції для командного адміністрування та більшої безпеки (таблиця 1.3).

Todoist підходить для тих, хто хоче зручно організувати свої завдання та проєкти, будь то для особистих потреб або для командної роботи, забезпечуючи ефективність та простоту використання.

Таблиця 1.3 - Порівняння цінових планів Todoist

Тарифний план	Вартість	Основні можливості
1	2	3
Beginner (Безкоштовний)	$0	5 активних проєктів, 5 співпрацівників на проєкт, 5 МБ для файлів, 3 фільтри, історія активності за 1 тиждень.
Pro	$4/місяць (при оплаті за рік)	300 проєктів, 25 співпрацівників на проєкт, 100 МБ для файлів, 150 фільтрів, нагадування, необмежена історія, теми та автоматичні резервні копії.

Кінець таблиці 1.3

1	2	3
Business	$6/місяць за користувача (при оплаті за рік)	Усі можливості Pro, плюс: 500 проєктів на учасника, 50 осіб на проєкт, командний інбокс, адміністрування доступу, централізоване управління виставленням рахунків.

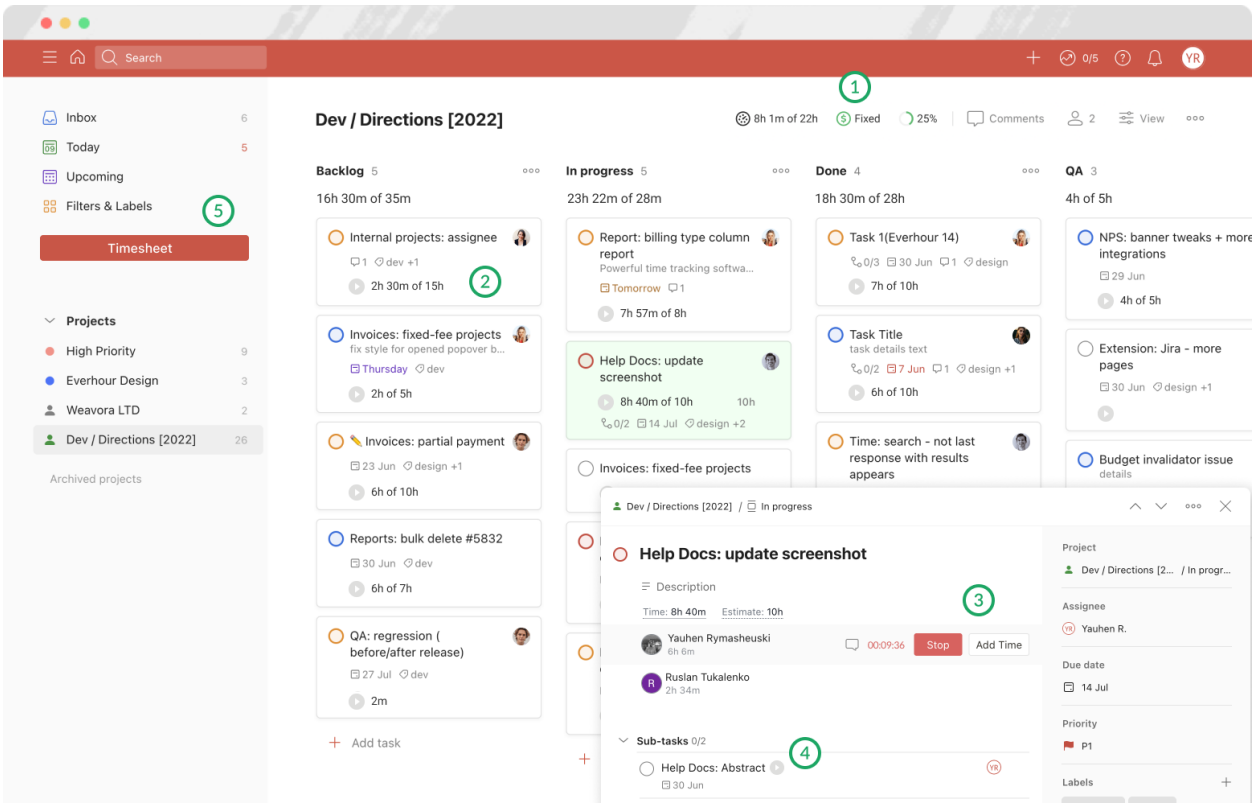

Рисунок 1.3 - Приклад застосунку Todoist

1.3.4 Jira

Jira - це система управління проєктами, розроблена компанією Atlassian, широко використовувана для відстеження помилок і управління завданнями в процесі розробки програмного забезпечення. З моменту свого першого випуску в 2002 році, Jira значно розширила свій функціонал і тепер застосовується в різних галузях, включно з маркетингом, фінансами та управлінням персоналом (рисунок 1.4) [17].

Система підтримує методології Scrum і Kanban, надаючи гнучкі інструменти для планування і моніторингу завдань. Jira доступна в хмарній версії (Jira Cloud) і серверній версії (Jira Server), з можливістю інтеграції з різними сервісами, такими як Adobe, Slack і Gmail.

У Jira завдання можна організовувати на дошках Scrum або Kanban, з можливістю додавання описів, чек-листів, файлів, призначення виконавців і спостерігачів, а також використання міток, підзадач і коментарів. Система пропонує високий ступінь налаштування інтерфейсу і робочого процесу, включно з автоматизацією завдань, інтеграцією з різними сервісами і розширеними функціями аналітики (таблиця 1.4).

Таблиця 1.4 - Порівняння цінових планів Jira

Тарифний план	Вартість	Основні можливості
1	2	3
Free	Безкоштовно	Для команд до 10 осіб, 2 ГБ сховища, доступ до базових функцій, підтримка спільноти.
Standard	$7,50	Для команд до 35 000 користувачів, 250 ГБ сховища, можливість анонімного доступу до проєктів, базові функції та доступ до додаткових прав.

Кінець таблиці 1.4

1	2	3
Premium	$14,50	Необмежене сховище, автоматизація завдань, розширена аналітика, для команд до 35 000 користувачів.
Enterprise	Індивідуальна ціна	Персоналізовані умови, підтримка до 50 000 користувачів, додаткові функції безпеки та управління.

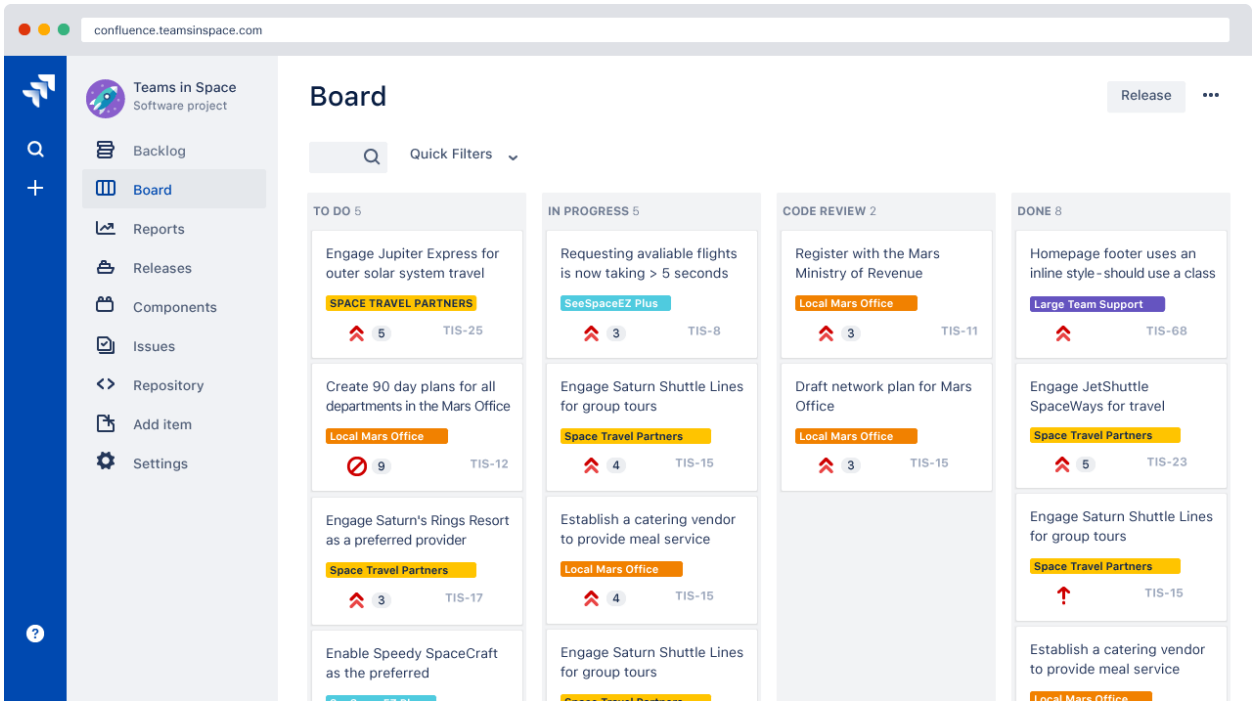

Рисунок 1.4 - Приклад застосунку Jira

1.3.5 Teamwork

Teamwork - це платформа для управління проєктами, яка була заснована у 2007 році. Вона орієнтована на командну роботу та підходить як для малого бізнесу, так і для великих компаній. Основна мета Teamwork - зробити процеси

планування, контролю та комунікації в проєктах більш прозорими та ефективними. Сервіс дозволяє створювати завдання, призначати виконавців, встановлювати дедлайни, додавати коментарі, обмінюватися файлами та стежити за прогресом у зручному вигляді (рисунок 1.5) [18].

Однією з головних переваг є гнучкість. Teamwork підтримує різні підходи до управління завданнями, зокрема роботу зі списками завдань, діаграмами Ганта, Канбан-дошками та календарями. Це дозволяє команді обирати той спосіб організації, який найкраще відповідає стилю її роботи. Крім того, платформа інтегрується з популярними сервісами, як-от Slack, Google Drive, Microsoft Teams, Dropbox та іншими. Це спрощує обмін даними між інструментами, які вже використовуються в компанії (таблиця 1.5).

Таблиця 1.5 - Порівняння цінових планів Teamwork

Тарифний план	Вартість	Основні можливості
1	2	3
Free Forever	Безкоштовно	Створення завдань (у вигляді списку, Ганта, календаря, Канбану), до 2 проєктів, створення команд, завантаження файлів до 100 МБ
Deliver	$10.99 / користувач / місяць (оплата щорічно)	До 300 проєктів, завантаження файлів до 100 ГБ, розширені інтеграції (Slack, MS Teams тощо), управління рахунками, командний чат
Grow	$19.99 / користувач / місяць (оплата щорічно)	До 600 проєктів, завантаження файлів до 250 ГБ, фінансове планування, звітність, спеціальні поля, історія змін, теги на рівні проєктів

Кінець таблиці 1.5

1	2	3
Scale	$54.99 / користувач / місяць (оплата щорічно)	Необмежена кількість проєктів, до 500 ГБ файлів, розширена аналітика, функції безпеки, підтримка клієнтів 24/7, планування ресурсів
Enterprise	Персональний розрахунок	Максимальна кастомізація, підтримка на рівні компанії, додаткові функції безпеки та контролю, можливість масштабування

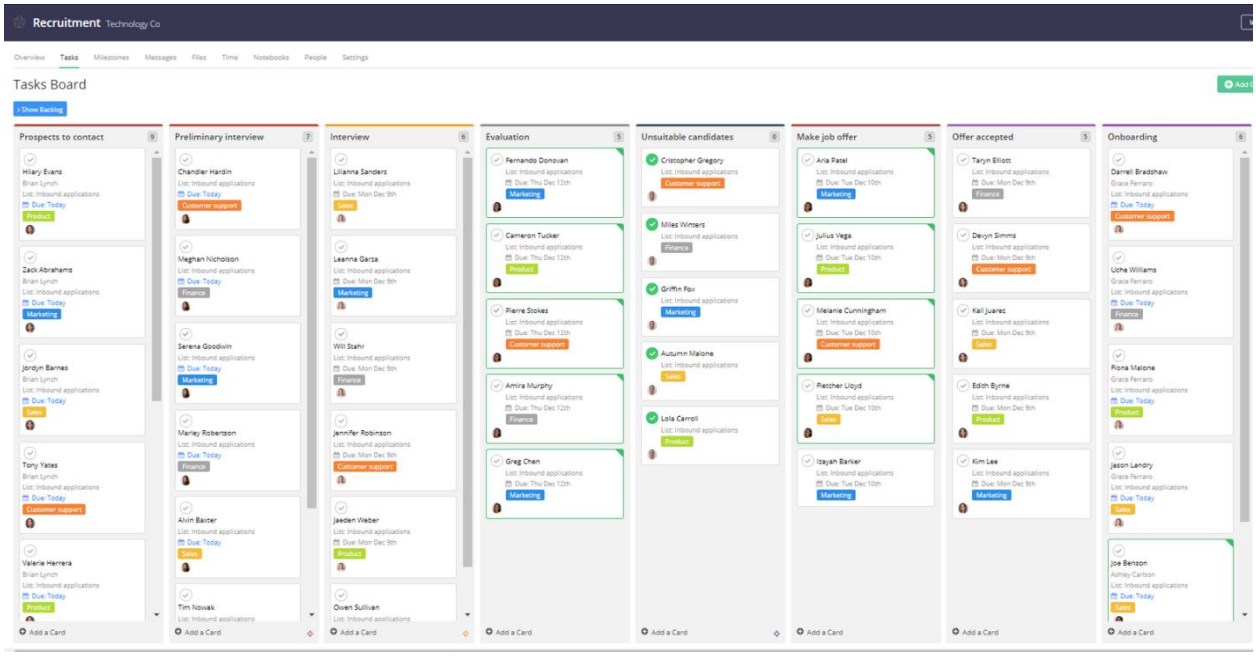

Рисунок 1.5 - Приклад застосунку Teamwork

У ході дослідження з’ясовано, що існує значна кількість сервісів для планування завдань у рамках проєктної діяльності. Щоб обрати найбільш відповідний інструмент для конкретної організації, варто насамперед враховувати специфіку її роботи. Водночас аналіз найпопулярніших сервісів показав, що найгнучкішим серед них є Jira. Це пояснюється широким функціоналом платформи, можливістю детального налаштування під

індивідуальні потреби, великою кількістю плагінів та потужними засобами для створення звітності. Важливо зазначити, що ринок онлайн-сервісів постійно оновлюється - з’являються нові рішення з розширеним функціоналом, тож варто регулярно моніторити їх розвиток.

1.4 Висновки за результатами аналізу та постановка задачі дослідження

За результатами проведеного аналізу гнучких методологій управління проєктами, таких як Scrum, Kanban, Extreme Programming, Lean, Scrumban та Crystal, а також платформ для управління проєктами, таких як Asana, Trello, Todoist, Jira та Teamwork, можна зробити кілька важливих висновків. Гнучкі методології, зокрема Scrum та Kanban, показали високу ефективність у швидко змінюваних та складних умовах розробки, оскільки вони дозволяють адаптувати процеси управління проєктами під поточні вимоги та забезпечують гнучкість у відповідь на зміни в вимогах і пріоритетах. Методології, такі як Extreme Programming, демонструють високу ефективність у командах, орієнтованих на якість коду та тісну взаємодію між розробниками та замовниками, однак потребують високого рівня професіоналізму та досвіду у керуванні процесами.

Що стосується платформ для управління проєктами, то інструменти, як-от Jira та Asana, надають широкі можливості для організації роботи над проєктами, зокрема мають розширений набір інструментів для планування, моніторингу та аналізу ходу виконання завдань, що робить їх підходящими для великих та складних проєктів. У той час як простіші платформи, як-от Trello та Todoist, є зручними для команд із меншою кількістю учасників або для задач, що не потребують складного управління, вони мають обмежену функціональність для масштабних проєктів.

Однак, незважаючи на численні переваги, гнучкі методології мають і певні обмеження. Зокрема, необхідність постійного моніторингу та вдосконалення процесів може вимагати додаткових ресурсів і зусиль. Це особливо помітно в методології Extreme Programming, де без належного досвіду можуть виникнути труднощі в управлінні технічними боргами та забезпеченні стабільної якості коду. Крім того, обмежена гнучкість деяких платформ для інтеграції з іншими інструментами може ускладнити процеси управління на великих і масштабних проєктах.

З огляду на ці висновки, основною метою даного дослідження є розробка та оцінка комбінованого підходу до управління проєктами, що інтегрує найкращі практики гнучких методологій для підвищення ефективності на всіх етапах життєвого циклу проєкту. У рамках дослідження буде здійснено порівняння ефективності використання Scrum та Kanban для управління проєктами в умовах веб-розробки з урахуванням специфіки команд та вимог до гнучкості. Також буде розроблено набір рекомендацій щодо оптимального вибору платформ для управління проєктами, що залежить від розміру команди та складності проєкту, а також визначено найбільш ефективні інструменти для інтеграції в рамках гнучких методологій. Окрім того, дослідження передбачає оцінку факторів, які впливають на успішну інтеграцію гнучких методологій та інформаційних систем у гібридні моделі життєвого циклу розробки програмного забезпечення, що сприятиме забезпеченню високої адаптивності та ефективності на всіх етапах проєкту.

2 ДОСЛІДЖЕННЯ ЗАСТОСУВАННЯ ГНУЧКИХ ГІБРИДНИХ МОДЕЛЕЙ ЖИТТЄВОГО ЦИКЛУ ДЛЯ ОКРЕМИХ РІЗНОВИДІВ ІТ-ПРОЄКТІВ

2.1 Порівняльний аналіз гнучких гібридних моделей життєвого циклу

Сучасна розробка програмного забезпечення потребує адаптивності, швидкого реагування на зміни вимог замовника та забезпечення високої якості продукту. Традиційні водоспадні моделі часто не задовольняють цим вимогам, тому дедалі більшої популярності набувають гнучкі (Agile) методології, які можна комбінувати, створюючи гібридні підходи. У цьому розділі буде проведено порівняльний аналіз шести поширених гнучких моделей: Scrum, Kanban, Extreme Programming, Lean, Scrumban та Crystal, з акцентом на їх особливості, переваги, недоліки та можливості поєднання.

2.1.1 Scrum

Тип життєвого циклу Scrum - інкрементальний, з ітеративним підходом, який організовує роботу в фіксовані цикли, що називаються спринтами, зазвичай тривалістю від одного до чотирьох тижнів. Кожен спринт має чітко визначений початок і кінець, а також конкретну мету і запланований обсяг роботи. Життєвий цикл Scrum передбачає проведення регулярних подій, які допомагають команді координуватися і контролювати прогрес. До таких подій належать планування спринту, де визначаються задачі на наступний період, щоденні короткі стендапи, що дозволяють обговорити поточний стан роботи, а також огляд результатів спринту і ретроспектива - аналіз процесу з метою його покращення.

Продукт розвивається за рахунок інкрементів, що поступово додають

користувацьку цінність. Основою організації роботи є Product Backlog - список всіх вимог і змін, які можуть виникати під час розробки. Scrum передбачає чітке розподілення ролей у команді. Product Owner відповідає за максимізацію цінності продукту, приймаючи рішення щодо пріоритетів і управління беклогом. Scrum Master допомагає команді правильно застосовувати фреймворк, усуваючи перешкоди та підтримуючи процес. Розробницька команда складається з крос-функціональних спеціалістів, які власне виконують основну роботу над продуктом.

Щодо технічних практик, Scrum не накладає жорстких вимог, тож для забезпечення високої якості часто застосовують додаткові методики, наприклад, практики XP або DevOps, які включають безперервну інтеграцію, автоматизоване тестування та інші інструменти підвищення стабільності коду. Гнучкість Scrum помітна, проте вона залежить від дисципліни команди. Цей фреймворк менш ефективний у ситуаціях із частими непередбачуваними змінами або за умов низької передбачуваності.

Метрики Scrum допомагають відслідковувати продуктивність і якість роботи. Найпоширенішими з них є Velocity, що показує кількість історій, завершених за спринт, Burndown Chart - графік, що ілюструє прогрес виконання задач, а також співвідношення обіцяного обсягу роботи і фактично завершеного (Commitment vs Done). До сильних сторін Scrum належить його ефективність для середніх та великих команд, здатність забезпечувати передбачуваність завдяки ітераціям, сприяння постійному вдосконаленню та простота масштабування. Водночас, цей підхід має недоліки, серед яких складність адаптації до частих змін вимог, висока залежність від компетенції Scrum Master і обмежена придатність для потокових, безперервних процесів підтримки.

Scrum рекомендовано застосовувати у проєктах, де зміни у вимогах відбуваються контрольовано, де є мотивована і крос-функціональна команда, а також де потрібна структурованість, прозорість процесу і регулярні поставки цінності замовнику.

2.1.2 Kanban

Kanban базується на безперервному потоці роботи без фіксованих ітерацій, які є у Scrum чи XP. У цьому життєвому циклі задачі поступово рухаються по стадіях, наприклад, від "To Do" до "In Progress" і "Done", і виходять після завершення. Такий підхід особливо корисний у проєктах із нерегулярним надходженням роботи або часто змінними пріоритетами. Головна ідея Kanban полягає у візуалізації робочого процесу на дошці та обмеженні кількості одночасно виконуваних задач, щоб не перевантажувати команду.

Задача проходить через кілька етапів - потрапляє у систему через беклог або запит клієнта, рухається по колонках, які можуть відповідати плануванню, розробці, тестуванню чи релізу, і завершується передачею замовнику або переходом у підтримку. Канбан дозволяє команді швидко реагувати на зміни, оскільки немає прив’язки до спринтів чи ітерацій, що робить його ідеальним для операційних, підтримуючих чи довготривалих продуктових проєктів.

Kanban не вводить нових ролей і зазвичай адаптується до існуючої організаційної структури, що значно спрощує його впровадження. Для роботи з Kanban важлива узгодженість у команді, але не потрібні формальні ролі типу Scrum Master чи Product Owner. Сам фреймворк не регламентує конкретних інженерних практик, але часто використовує інструменти, які допомагають керувати потоком: автоматичні сповіщення при зміні статусу задач, ліміти на кількість одночасних завдань у кожній колонці, встановлення очікуваного часу завершення задач (Service Level Expectations), а також моніторинг циклу виконання (Cycle Time) і часу очікування (Lead Time). Часто Kanban поєднують із технічними практиками XP або Lean, щоб підвищити якість і ефективність.

Гнучкість Kanban полягає у тому, що його можна вводити поступово без суттєвих змін у структурі або процесах. Зміни пріоритетів не вимагають

перепланування ітерацій - просто додаються нові задачі у чергу. Однак ця свобода іноді призводить до розмивання фокусу, якщо не встановити чіткі політики та обмеження.

Для оцінки процесу використовують метрики, які допомагають виявити вузькі місця і оптимізувати потік. Це час виконання задачі (Cycle Time), загальний час від запиту до завершення (Lead Time), кількість завершених завдань за певний період (Throughput) і візуалізація балансу навантаження на команду (Cumulative Flow Diagram).

Kanban вирізняється високою гнучкістю, простотою впровадження, прозорістю процесу та швидким виявленням проблем. Він підходить для команд із змінними обсягами задач і добре сумісний з будь-якою організаційною структурою. Однак відсутність ітерацій ускладнює прогнозування результатів, немає чіткої ролі, відповідальної за покращення процесу, можливе "перетягування" завдань без їх завершення, а також Kanban менш підходить для команд, які тільки починають працювати за Agile.

Використовувати Kanban варто, коли задачі надходять нерегулярно або без чіткого прогнозу, коли потрібно зберегти існуючу структуру команди без введення нових ролей, коли багато паралельної роботи і важливо оптимізувати потік, а також у тривалих чи постійних проєктах, таких як підтримка або DevOps.

2.1.3 Extreme Programming

Extreme Programming - це гнучка методологія розробки програмного забезпечення, яка робить акцент на технічній досконалості, високій адаптивності до змін та швидкому зворотному зв’язку. XP працює в коротких ітераціях тривалістю 1-2 тижні, протягом яких продукт розвивається через часті релізи, що приносять користь кінцевому користувачу.

Процес XP базується на тісному зворотному зв’язку на різних рівнях: із замовником для визначення пріоритетів функціоналу, із командою для обговорення архітектурних і технічних рішень, а також із самим кодом через постійне тестування, рецензії й інтеграцію. Ітерація включає планування, розробку, автоматизоване тестування, безперервну інтеграцію та часті релізи.

Команда XP має мінімальну кількість ролей: замовник, який присутній у команді і визначає пріоритети; програмісти - крос-функціональна команда, яка бере участь у всіх активностях; наставник (Coach), що допомагає впроваджувати практики XP; і Tracker, який відстежує метрики і продуктивність. Ролі в команді гнучкі, вся команда працює тісно і без поділу на технічних та нетехнічних учасників.

Основою XP є набір з 12 технічних практик, серед яких найважливішими є парне програмування, коли код пишеться двома розробниками одночасно, тестування через написання тестів перед кодом (TDD), постійна інтеграція змін із перевіркою, регулярне рефакторинг - поліпшення архітектури без зміни функціоналу, колективна відповідальність за код і підтримка сталого робочого ритму без перевантажень. Завдяки цьому XP допомагає створювати високоякісний продукт з мінімальним технічним боргом.

Щодо гнучкості, XP дозволяє замовнику змінювати, додавати або вилучати функціонал перед кожною ітерацією, що дуже корисно для динамічних ринків, стартапів і дослідницьких проєктів. Водночас високі вимоги до технічної майстерності можуть стати перепоною для команд без досвіду в парному програмуванні, TDD чи CI/CD.

Для оцінки процесу в XP використовують метрики, такі як velocity - кількість завершених користувацьких історій за ітерацію, покриття коду автоматизованими тестами, стабільність збірок у системі безперервної інтеграції, а також обсяг змін у коді, що відображає стабільність архітектури. Крім того, важливим є відстеження дефектів у продакшені та часу їх усунення.

Сильні сторони XP - це висока якість коду, швидке виявлення помилок завдяки частим релізам, гнучка реакція на зміну вимог, підтримка

інноваційних і ризикових проєктів, а також формування сильної командної культури. Однак XP вимагає високого технічного рівня команди, дисципліни, і не всі розробники легко приймають парне програмування. Крім того, методологія складно масштабується на великі розподілені команди.

XP є особливо ефективним у проєктах зі швидко змінними вимогами, де критично важлива висока якість коду, наприклад у банківській сфері, медицині або інших критичних системах. Він найкраще підходить для досвідчених команд, готових до практик TDD, парного програмування і безперервної інтеграції, а також у випадках, коли клієнт активно залучений до розробки.

2.1.4 Lean Software Development

Lean - це підхід, що бере початок у виробничій системі Toyota і адаптований до IT-сфери. Він орієнтується на максимальне створення цінності для клієнта за мінімізації втрат і не задає жорсткого циклу розробки. Замість цього Lean передбачає гнучкий потік роботи, у якому продукт розвивається інкрементально, але без суворої ітеративності. Основна увага приділяється ефективності й швидкості поставки.

Процес у Lean виглядає як вільний потік завдань, що проходять через різні стадії з обмеженням на кількість одночасно виконуваної роботи, подібно до Kanban. Проте головний акцент робиться не лише на потоковість, а й на оптимізацію всієї системи цілком. Принципи Lean включають видалення втрат, підвищення якості, створення знань, відкладання прийняття рішень до останнього можливого моменту, швидку поставку, повагу до людей і оптимізацію системи в цілому.

Щодо ролей, Lean не встановлює суворих формальних ролей, але підкреслює важливість залучення кожного учасника та повагу до його

експертизи. Зазвичай у команді присутній Lean Coach або системний мислитель, який допомагає виявляти «вузькі місця» та оптимізувати процес. Розробники є багатофункціональними й автономними, а бізнес-замовник забезпечує пріоритети та зворотній зв’язок з клієнтами. Ключовим є не те, хто виконує завдання, а як ефективно організований процес поставки цінності.

Lean не нав’язує конкретних технічних практик, але підтримує ті, що сприяють мінімізації втрат і підвищенню якості. Серед таких підходів - аналіз втрат (Value Stream Mapping), впровадження мікросервісів, CI/CD, DevOps для швидкого зворотного зв’язку і поставки, а також тестування з автоматизацією і системне логування з аналізом корінних причин помилок. Lean є, перш за все, мисленням, тому технічні рішення мають змінюватися відповідно до контексту.

Щодо гнучкості, Lean відзначається високою адаптивністю, але фокусується на системному мисленні. Це означає, що процеси, інструменти і навіть архітектуру можна змінювати, якщо це допомагає зменшити втрати і прискорити доставку цінності. Зміни мають бути обґрунтованими та перевірятися через експерименти, реалізовані за циклом build-measure-learn.

Для оцінки ефективності Lean активно використовує системні метрики, такі як час проходження завдання (Cycle Time), час від отримання завдання до його виконання (Lead Time), кількість виконаних завдань за період (Throughput), ефективність потоку цінності (Value Stream Efficiency), а також обсяг роботи, що виникає через помилки (Failure Demand). Важливу роль відіграють візуальні інструменти, зокрема діаграма кумулятивного потоку (Cumulative Flow Diagram), що допомагає виявляти вузькі місця. Окрім технічних метрик, Lean акцентує увагу на бізнес-показниках, таких як швидкість доставки, рівень прийняття продукту користувачами, час виходу на ринок і повернення інвестицій.

Сильні сторони Lean полягають у гнучкості на рівні всієї системи, а не лише окремих задач, у здатності виявляти та усувати системні втрати, підтримці культури постійного вдосконалення, орієнтації на бізнес-результат

і добрій сумісності з іншими фреймворками, наприклад, Kanban, Scrum чи DevOps. Проте Lean вимагає стратегічного мислення і дисципліни, складний для впровадження без підтримки керівництва, може здатися розмитим для команд, що звикли до чітких правил, і потребує часу на аналіз та оптимізацію системи.

Lean найкраще підходить для випадків, коли потрібно мінімізувати час від ідеї до поставки, існують високі вимоги до ефективності й усунення зайвих витрат, команда досить зріла та готова до самостійної оптимізації, а також для довготривалого розвитку продукту чи великої системи.

2.1.5 Scrumban

Scrumban - це гібридна методологія, яка поєднує Scrum і Kanban, створюючи ітеративно-потоковий життєвий цикл. Вона об'єднує структурованість ітерацій Scrum з гнучкістю та візуалізацією потоку роботи, характерними для Kanban. У Scrumban команда планує роботу в регулярних циклах, зазвичай раз на один-два тижні, але при цьому не обмежена жорсткими спринтами, що дає можливість гнучко підлаштовувати обсяг задач під поточні потреби.

Процес передбачає використання Kanban-дошки з лімітами на кількість задач у роботі (WIP), а також легкі ритуали, які замінюють класичні Scrum-церемонії. Зустрічі для планування проходять регулярно, щоденні стендапи можуть бути опціональними, а ретроспективи проводяться за потребою або після накопичення змін. Такий підхід дозволяє команді швидше реагувати на зміни, не втрачаючи контролю над процесом.

Щодо ролей, Scrumban не накладає строгих вимог, але часто зберігає структуру Scrum: Product Owner визначає пріоритети задач, команда бере на себе відповідальність за їх виконання, а Scrum Master або Delivery Manager

виконує роль фасилітатора. Іноді ця роль може зникати, якщо команда достатньо самоорганізована.

Хоча Scrumban сам по собі не встановлює технічних практик, він часто поєднується з XP або Lean підходами, зокрема з автоматизованим тестуванням, CI/CD, контролем часу виконання задач і обмеженнями на кількість паралельної роботи. Планування здійснюється “just-in-time”, тобто лише тоді, коли це потрібно, що дозволяє зберігати гнучкість.

Гнучкість Scrumban полягає в тому, що команда може змінювати обсяг задач у процесі ітерації і не обмежена тривалістю спринту. Це дає змогу поступово налаштовувати процес під конкретні потреби без різких змін.

Щодо метрик, у Scrumban використовуються класичні показники Kanban і Scrum - Lead Time, Cycle Time, рівень задач у роботі (WIP), кількість завершених задач (Throughput), заблоковані задачі і прогнозування, наприклад, за допомогою Monte Carlo Simulation. Для приблизного планування іноді зберігають і velocity.

До сильних сторін Scrumban належать висока гнучкість, менше ритуалів у порівнянні зі Scrum, прозорість процесу завдяки Kanban-дошці, можливість поступового впровадження ідеї без кардинальних змін, що робить його зручним для стабільних і досвідчених команд, які прагнуть розвиватися. Проте він вимагає дисципліни і самоорганізації, і без належного контролю процес може стати хаотичним. Команди без досвіду ризикують втратити фокус, а менеджменту іноді складно прийняти менш формалізований підхід.

Scrumban добре підходить для команд, які вже працювали зі Scrum або Kanban і хочуть поєднати найкраще з обох світів, для проєктів із частими змінами пріоритетів, де потрібна гнучкість і зниження ритуальності, але збереження контролю. Особливо корисний для довготривалих проєктів із мінливими цілями і вимогами.

2.1.6 Crystal

Тип життєвого циклу Crystal - ітеративний та адаптивний. Це сімейство легких гнучких методологій, створене Алістером Коберном, яке базується на ідеї, що не існує універсального процесу для всіх проєктів. Методологія підлаштовується під розмір команди, критичність проєкту й рівень складності комунікацій. Crystal має кілька варіантів, таких як Crystal Clear для невеликих команд до 6 осіб, Crystal Orange - для команд середнього розміру, і Crystal Red - для критичних систем.

Підхід Crystal мінімалістичний і орієнтований на людей і взаємодію, які цінуються вище за процеси та інструменти. Важливими є регулярне вдосконалення, часті поставки робочого програмного забезпечення та інкрементальна розробка з циклами тривалістю 2-4 тижні. Процес передбачає оцінку критичності проєкту, вибір відповідного варіанту методології, мінімальну документацію, активну комунікацію (іноді команда працює фізично разом) і часті поставки з рефлексією над результатами.

Ролі у Crystal не формалізовані, і команда самостійно обирає структуру відповідно до своїх потреб. Зазвичай у проєкті є розробники з високою автономією, замовники, які перевіряють результати, координатор або фасилітатор для узгодження, а в складніших випадках - тестувальники, дизайнери чи архітектори. Такий підхід підтримує самоорганізацію і крос-функціональність без зайвої бюрократії.

Crystal не вимагає обов’язкових технічних практик, але заохочує використання автоматичного і інтеграційного тестування, рефакторингу, інкрементальної архітектури, проведення postmortem-сесій і дизайн через код у менш формальних проєктах. Головний акцент - на простоті і достатності інструментів, а не на їх суворому дотриманні.

Ця методологія відзначається високою гнучкістю і здатністю адаптуватися до специфіки кожного проєкту. Команда сама вирішує, які

практики і метрики використовувати, що дозволяє швидко реагувати на зміни у пріоритетах або вимогах. Crystal особливо підходить для команд, які прагнуть уникнути зайвої "агільної бюрократії".

Метрики Crystal не стандартизовані, але часто застосовують такі показники, як швидкість виконання ітерацій (velocity), частоту релізів, час отримання зворотного зв’язку від клієнтів, загальний час циклу розробки, кількість помилок у продакшені та відстеження дій за підсумками ретроспектив. Головна їхня мета - підтримка постійного вдосконалення, а не створення бюрократичних звітів.

До сильних сторін методології належать висока адаптивність, підтримка команд будь-якого розміру, мінімум процесного навантаження, акцент на комунікаціях і якості, а також придатність для довгострокових і внутрішніх проєктів. Однак відсутність чітких інструкцій ускладнює впровадження для новачків, існує ризик анархії без достатнього досвіду, потрібна культура самодисципліни і відповідальності, і методологію важко "продати" керівництву, яке звикло до жорстких структур.

Crystal найкраще підходить для досвідчених самоорганізованих команд, яким потрібна гнучкість і мінімум формальностей, а також для внутрішніх проєктів без тиску зовнішніх стандартів, де важливі якісна взаємодія і довіра в команді.

2.2 Порівняльний аналіз різновидів IT проєктів

У сфері інформаційних технологій існує велике різноманіття типів проєктів, кожен з яких має свою специфіку, вимоги до управління, життєвий цикл, рівень ризику, технологічну базу та цілі. Успішне управління IT-проєктами неможливе без чіткого розуміння їх природи. У цьому розділі детально проаналізуємо основні типи IT-проєктів: продуктові, аутсорсинг-

проєкти, R&D, підтримка, внутрішні корпоративні проєкти. Для кожного з них визначено призначення, особливості, переваги, ризики, а також відповідні підходи до управління та приклади застосування.

2.2.1 Продуктові проєкти

Продуктові проєкти спрямовані на створення та розвиток власного продукту, який компанія напряму пропонує ринку. Це можуть бути мобільні застосунки, SaaS-платформи, ігрові сервіси або CRM-системи. Компанія виступає власником продукту та несе відповідальність за його візію, розвиток, монетизацію і успіх на ринку. Головна мета - задовольнити кінцевого користувача і забезпечити комерційну життєздатність продукту.

Особливості таких проєктів полягають у фокусі на користувачі, постійному ітеративному розвитку, орієнтації на зворотний зв’язок з ринку, монетизації та масштабуванні. Інновації, UX-дослідження, A/B тестування і розробка на основі гіпотез відіграють ключову роль, а продуктова гнучкість дозволяє змінювати функції і цілі залежно від реакції ринку.

Команда зазвичай має крос-функціональний склад, у якому присутні Product Owner або Product Manager, що визначає візію та бізнес-пріоритети, Tech Lead або Architect, UI/UX дизайнер, розробники фронтенду, бекенду і мобільних додатків, QA-інженери, аналітики даних і, іноді, маркетингова або growth-команда. Робота ведеться за методологіями Scrum, Kanban, Scrumban або Lean Startup, а іноді - їх комбінаціями.

Розробка починається з визначення проблеми користувача та створення мінімально життєздатного продукту (MVP). Після релізу збирають метрики, аналізують їх і адаптують продукт. Ітерації включають вдосконалення або відмову від функцій, масштабування ефективних рішень і постійне UX-тестування з експериментами. Іноді може знадобитися повний перезапуск

(pivot). Цикл розробки базується на feedback loops: Build → Measure → Learn.

Технічно проєкти підтримують швидкі релізи через CI/CD, використання feature toggles і A/B тестування для безпечного впровадження змін. Інфраструктура має бути масштабованою, з аналітикою, моніторингом і високими вимогами до UX та продуктивності. Часто застосовують Data-driven підхід і Domain-Driven Design для складних систем.

Ключові метрики успіху включають активність користувачів (DAU/WAU/MAU), показники утримання (Retention Rate), конверсії, життєву цінність клієнта (CLTV), доходи і відтік (churn), задоволеність користувачів (CSAT/NPS), час виходу на ринок і успішність A/B тестів. Вони мають відображати не тільки технічну якість, а й бізнес-результати та користь для користувача.

Переваги продуктового підходу - це велика гнучкість у виборі напряму, безперервний розвиток, можливість масштабувати успішні ідеї, фокус на користувача для формування якісного UX і спільний ріст команди разом із продуктом. Водночас, цей тип проєктів супроводжується високою невизначеністю, потребує глибокої продуктової експертизи і важко відмовлятися від невдалих ідей, що вже «коштували зусиль». Конкуренція висока, і продукт має вирізнятися, а команда може відчувати вигорання через постійні експерименти.

Продуктові проєкти підходять стартапам і компаніям з власною ідеєю і ринком, коли потрібно знайти Product-Market Fit, для інноваційних задач, що вимагають швидкої адаптації, і при наявності гнучкого бюджету та чіткого внутрішнього бачення продукту.

2.2.2 Аутсорсинг-проєкти

Аутсорсингові проєкти - це замовлення, які компанія виконує для іншого бізнесу, включаючи розробку, підтримку, модернізацію або надання фахівців у команду клієнта. Клієнт зазвичай володіє баченням, кінцевими вимогами і визначає остаточне рішення щодо релізу. Виконавець повинен якісно, вчасно і в межах бюджету надати результат, що задовольняє вимоги замовника.

Основна специфіка таких проєктів полягає в роботі з зовнішнім замовником у форматі B2B, за фіксованими або погодинними бюджетами. Клієнт контролює процес, виставляє чіткі вимоги та дедлайни, а виконання орієнтується на контрактні зобов'язання і SLA, з обов'язковими звітами, регулярними демо і погодженнями функцій.

Команда зазвичай включає Project Manager, який координує роботу з клієнтом і веде контракт, Business Analyst, що формалізує вимоги, Tech Lead або Architect, відповідальних за технічні рішення, а також розробників, QA, DevOps та фінансового менеджера. З боку клієнта часто присутній власний Product Owner або його делегат. Співпраця в проєкті базується на двосторонній комунікації між командами виконавця і клієнта.

Розробка проходить за моделлю співпраці - Time & Material, Fixed Price або Dedicated Team. Вона включає збір і формалізацію вимог, оцінку і планування (зазвичай у вигляді WBS або спринт-плану), власне розробку з використанням Scrum, Kanban або Water-Scrum-Fall, QA, підтримку, а також регулярне звітування замовнику про прогрес, ризики і демонстрації. Після завершення проходить приймальне тестування і запуск.

Технічні аспекти проєкту залежать від вимог клієнта щодо стеку, інфраструктури та безпеки. CI/CD налаштовується за узгодженням, часто виникає необхідність документування кожної зміни. Для великих проєктів важливі DevSecOps і аудит. Іноді доступ до середовищ розмежовується або

частково використовуються ресурси клієнта, що обмежує гнучкість технічних рішень.

Успіх проєкту оцінюють за виконанням бюджету і термінів, ступенем виконання плану, кількістю змін у вимогах, задоволеністю клієнта (NPS), рівнем дефектів до продакшну, витратами на команду і метриками SLA, такими як час реакції та вирішення інцидентів.

Сильними сторонами аутсорсингу є стабільне фінансування з гарантованим бюджетом, можливість масштабування через додаткові команди, формалізація вимог, що знижує ризики, стандартизовані процеси та швидкий старт без потреби у створенні власної візії продукту. Проте свобода у технічних рішеннях обмежена, а зміни вимог можуть викликати бюрократію і конфлікти. Довгі погодження та формальні комунікації іноді гальмують роботу, відсутній прямий контроль над кінцевим користувачем, і команда може втрачати мотивацію через обмежений вплив на кінцевий результат.

Аутсорсинг застосовують, коли клієнт не має власної R&D-команди або прагне знизити витрати, для корпоративних, державних чи інших зовнішніх замовлень, а також у випадках, коли потрібно швидко реалізувати чітко визначений функціонал і проєкт має конкретні рамки та цілі.

2.2.3 R&D-проєкти (Research and Development Projects)

R&D-проєкти спрямовані на дослідження нових ідей, технологій або інноваційних концепцій. Їх основна мета - перевірка гіпотез, створення прототипів і вивчення технічної здійсненності. Це високоризикові, але потенційно високоприбуткові ініціативи з невизначеним результатом, які можуть дати проривний ефект.

Основні характеристики:

- фокус на дослідженні, не на релізі;

- гіпотези, експерименти, прототипи (PoC, MVP);
- висока технічна складність і новизна;
- швидкі ітерації і навчання;
- часто обмежене грантове або венчурне фінансування.

Ролі в команді:

- R&D Lead / Innovation Manager - формує напрямок дослідження;
- Tech Experts / Engineers - розробляють прототипи;
- Data Scientists / ML Researchers - за потреби, в AI-проєктах;
- Product Explorer / Strategist - збирає потреби ринку;
- UX-дослідники та аналітики - допомагають перевіряти ідеї;
- Можливе залучення науковців та партнерів.

Процес розробки:

- формулювання цілей і гіпотез;
- створення експериментального середовища;
- швидка ітерація над прототипами;
- збір і аналіз даних;
- закриття невдалих напрямків і масштабування успішних;
- документація результатів (наукові звіти, whitepaper).

Технічні аспекти включають використання експериментальних технологій, різнорідних середовищ (cloud, hardware, AI), швидке створення прототипів без повної архітектури, наукову валідацію, тестування гіпотез та часткову або відсутню DevOps-підтримку.

Метрики успіху: кількість перевірених гіпотез, час до першого прототипу, технічна новизна, частка рішень, що перейшли у продакшн, патенти і публікації, ROI від впроваджень.

Сильні сторони R&D - генерація інновацій, стимул для творчості, свобода експериментів, основа для майбутніх продуктів. Слабкі - високий ризик провалу, складність оцінки короткострокового ефекту, труднощі планування, можливе втрачання фокусу.

R&D доцільно застосовувати для лідерських технологічних компаній,

при потребі дослідити нові технології, створити PoC, в довгостроковій продуктовій стратегії, а також у сферах AI, AR/VR, Blockchain, Quantum тощо.

2.2.4 Підтримка та обслуговування

Support-проєкти зосереджені на підтримці вже існуючих систем: виправлення помилок, оптимізація продуктивності, моніторинг, оновлення, адаптація під нові умови (наприклад, законодавчі зміни чи нові API). Вони забезпечують стабільність, безперервність і якість функціонування ПЗ, часто є частиною SLA з клієнтом або внутрішньої IT-служби.

Основні характеристики таких проєктів: робота з legacy або production-системами, обов'язкові регламенти та процедури ескалації, постійний моніторинг і регулярні оновлення, реакція на інциденти, баги, регресії, довготривалий життєвий цикл. Можуть містити незначні нові функції, але головний фокус - на стабільності.

Ключові ролі в команді:

- Support Lead / Team Lead - координує роботу і розподіляє запити;
- DevOps / SRE - відповідає за інфраструктуру і моніторинг;
- Support Engineers / Fix Team - виправляють баги;
- QA / Regression Testers - контролюють стабільність змін;
- Service Desk / Dispatcher - приймають запити і підтримують користувачів;
- Account / Service Manager - відповідають за комунікацію з клієнтом і SLA.

Процес підтримки включає обробку інцидентів з Service Desk, аналіз логів і діагностику, виправлення (іноді екстрені hotfix), тестування і rollback-плани, реліз у вікна низької активності, звітування за SLA (час реакції та вирішення). Управління ведеться через системи типу Jira Service Management

або ServiceNow з інтеграцією CI/CD та моніторингом.

Технічні аспекти:

- робота з legacy-кодом і великою кодовою базою;
- використання систем моніторингу (Grafana, Prometheus, ELK);
- автоматизація перевірок, інцидент-менеджмент;
- hotfix, rollback, zero-downtime deployments;
- мінімальні зміни з максимальною безпекою;
- DevSecOps та SOC для критичних проєктів (банки, телеком).

Успішність оцінюється за SLA-показниками (час реакції і вирішення), MTTR, кількістю повторних інцидентів, Change Success Rate, uptime системи, кількістю критичних помилок у продакшні та backlog-ом багів.

Support-проєкти забезпечують стабільність бізнесу, мають чіткі процеси і прозору звітність, їх ефективність легко оцінити. Водночас, через монотонність і тиск SLA мотивація команди може знижуватись, а робота з legacy-системами ускладнюється відсутністю документації та обмеженими можливостями для інновацій.

Такі проєкти доцільно застосовувати після релізу продукту, при відповідальності за SLA або гарантійне обслуговування, для систем із високою критичністю (медицина, банки, телеком), коли стабільність важливіша за новий функціонал і якщо система активно використовується без планів заміни.

2.2.5 Внутрішні корпоративні проєкти

Внутрішні корпоративні IT-проєкти, або Internal IT (Enterprise Projects), реалізуються всередині компанії для власних потреб, а не для зовнішнього клієнта чи відкритого ринку. Їхня мета - покращити ефективність, автоматизувати процеси, підвищити прозорість або безпеку. Це можуть бути CRM, ERP, HRM-системи, внутрішні портали, звітність, документообіг,

аналітика, фінансові інструменти та інші рішення.

Основними замовниками таких проєктів є внутрішні підрозділи компанії, наприклад HR, фінанси, юристи чи безпека. Процеси зазвичай формалізовані, включаючи тендери, узгодження і бюджетування. Часто в проєкті залучена велика кількість зацікавлених сторін, а він інтегрується з багатьма внутрішніми системами та застосовується у цифровій трансформації. Хоча прямий ROI обмежений - проєкт не продається зовнішньому ринку, він оптимізує внутрішні витрати. Внутрішні проєкти можуть бути великими і багаторічними або короткими ініціативами під конкретні потреби.

У команді традиційно присутні Internal Product Owner або Business Sponsor, що представляють внутрішнього замовника, Business Analyst, який перекладає потреби користувачів у вимоги, Enterprise Architect або Integration Lead, відповідальний за зв’язки з іншими системами, а також Delivery Manager або PM, який координує строки, ресурси та бюджети. Розробники, QA та DevOps реалізують систему, а Key Users - внутрішні користувачі - проходять UAT та впливають на прийняття рішень. Також залучаються представники інфраструктури, безпеки та compliance, оскільки проєкт розгортається всередині організації.

Процес розробки включає збір вимог і затвердження бізнес-кейсів, формалізацію бюджету з узгодженням IT-департаментом, архітектуру та інтеграції з корпоративними системами. Реалізація може відбуватися за Agile, Waterfall або гібридними підходами. Після UAT система переходить у Go-Live та передається на підтримку внутрішній IT-службі. Ці проєкти часто перебувають під внутрішнім контролем, включаючи аудит та звітність.

Технічні аспекти передбачають інтеграцію з LDAP, Active Directory, корпоративними порталами, а також використання MS SharePoint, SAP, 1C, Jira, Odoo або low-code платформ. Важливі суворі вимоги до автентифікації, шифрування та резервного копіювання, а також обмеження на cloud або зовнішні API згідно з політиками безпеки. DevOps-практики застосовуються частково, оскільки релізи часто потрібно узгоджувати, а документація і

тестування суворо регламентовані.

Основні метрики успіху проєктів - це дотримання бюджету та строків, задоволеність внутрішніх користувачів (CSAT, NPS), кількість автоматизованих процесів, час виконання операцій до і після впровадження, зменшення ручної роботи та помилок, інтеграційна сумісність із іншими системами, а також зниження витрат або підвищення ефективності.

Серед сильних сторін - безпечне середовище та передбачуване замовлення, можливість глибокого вбудовування у наявні процеси, значний вплив на всю організацію, простір для комплексної архітектури і довгострокових рішень, а також накопичення корпоративної експертизи. Проте є і слабкі сторони: складність узгоджень із різними департаментами, політичні або неформальні пріоритети, часта зміна вимог через внутрішні процеси, обмежений інтерес розробників через відсутність ринку і «неплатних» користувачів, а також ризик затягування або «похованих» ініціатив.

Такі проєкти доцільно застосовувати, коли потрібно автоматизувати або покращити внутрішні бізнес-процеси, реалізовувати цифрову трансформацію в великій компанії, за наявності незалежного IT-бюджету чи програми покращення ефективності, в межах корпоративної стратегії розвитку або безпеки, а також коли зовнішні системи не підходять або надто дорогі.

2.3 Розробка критеріїв для оцінювання вибору моделі гнучкої гібридної моделі життєвого циклу для конкретного різновиду IT проєктів

У процесі управління IT-проєктами одним із ключових етапів є обґрунтований вибір моделі життєвого циклу, яка б відповідала характеристикам конкретного проєкту. З огляду на широке розмаїття типів IT-проєктів (продуктові, аутсорсингові, дослідницько-інноваційні (R&D),

проєкти технічної підтримки, внутрішні корпоративні ініціативи), виникає необхідність у системному підході до зіставлення потреб проєкту з можливостями різних гнучких гібридних моделей.

З цією метою було розроблено набір формалізованих критеріїв, що дозволяють оцінити рівень відповідності певної гібридної методології життєвого циклу потребам конкретного типу проєкту. Кожен критерій оцінює той чи інший аспект проєктного контексту, що має значення для вибору методологічного підходу.

До ключових критеріїв оцінювання було віднесено:

− визначеність вимог - наскільки чітко сформульовані функціональні й нефункціональні вимоги на початку проєкту.

− рівень залучення замовника - інтенсивність взаємодії із зацікавленими сторонами протягом розробки.

− масштаб проєкту - розмір команди, кількість модулів, складність інтеграцій.

− тривалість виконання - загальний очікуваний період реалізації проєкту.

− критичність помилок - допустимість помилок та їх вплив на бізнес.

− інноваційність задачі - ступінь новизни, необхідність досліджень, експериментів.

− потреба в інтеграції з іншими системами - наявність зовнішніх або внутрішніх залежностей.

− наявність регуляторних вимог - дотримання стандартів, політик безпеки, норм відповідності.

− зрілість команди - досвід роботи в гнучких командах, самостійність, технічна експертиза.

− частота змін вимог - схильність проєкту до змін у специфікаціях протягом реалізації.

Для забезпечення об’єктивного зіставлення між вимогами різновидів IT-проєктів та властивостями гнучких методологій (Scrum, Kanban, XP, Lean,

Scrumban, Crystal), кожен тип проєкту та кожну методологію було формалізовано у вигляді бінарного вектора, де кожна позиція відповідає одному з десяти узагальнених критеріїв. Значення 1 означає високу релевантність або підтримку певного критерію, а 0 - його відсутність або низьку значущість.

На основі цих векторів були створені таблиці відповідності для кожного з типів проєктів - продуктових, аутсорсингових, R&D, підтримки та внутрішніх корпоративних. У кожній таблиці представлено, наскільки кожна з методологій відповідає визначеним критеріям для конкретного типу проєкту.

Для кількісного оцінювання відповідності методології до проєкту було застосовано відстань Хеммінга - метрику, яка обчислює кількість невідповідностей між вектором потреб проєкту та вектором можливостей методології. Чим менша відстань Хеммінга, тим вища ступінь відповідності обраної гнучкої моделі потребам певного типу проєкту.

Для обчислення відстані Хеммінга використовується дуже проста формула:

$$D_H(A,B) = \sum_{i=1}^{n}[A_i \neq B_i] \quad (1.1)$$

де: D_H(A,B) - відстань Хеммінга між двома бінарними векторами A і B,

A_i, B_i- значення у і-ій позиції векторів A та B,

$[A_i \neq B_i]$ - індикаторна функція, яка дорівнює 1, якщо значення різні, і 0, якщо однакові,

n - довжина вектора (тобто кількість критеріїв у нашому випадку).

Приклад:

Вектор проєкту: [0, 1, 1, 1, 1, 1, 1, 0, 1, 1]

Вектор методології: [1, 1, 0, 1, 1, 1, 0, 1, 0, 1]

Тоді:

$$D_H=1+0+1+0+0+0+1+1+1+0==5$$

Тобто, вектора відрізняються у 5 позиціях - отже, відстань Хеммінга дорівнює 5.

Такий формалізований підхід дозволяє не лише візуально порівнювати

моделі у таблицях, а й приймати обґрунтовані рішення на основі кількісного аналізу. Це особливо важливо для великих організацій, де одночасно реалізуються різнорідні проєкти з відмінними вимогами до управління, інноваційності, інтеграції та швидкості змін.

2.3.1 Продуктові проєкти

Продуктові проєкти зазвичай орієнтовані на створення унікального ринку або рішення з високим рівнем інноваційності, частими змінами вимог, ітеративним розвитком функціональності та тривалим життєвим циклом. У таких умовах критично важливою є здатність команди швидко адаптуватися до нових викликів, підтримувати стабільну якість продукту та забезпечувати зрозумілу документацію для супроводу.

Для оцінки релевантності методологій до потреб продуктових проєктів було сформовано ідеальний вектор, який відображає ключові вимоги до організації роботи над продуктом. Цей вектор має вигляд:

[0,1,1,1,1,1,1,0,1,1]

Кожна позиція у векторі відповідає одному з десяти визначених критеріїв, таких як частота змін, інноваційність, автономія команди, наявність ітерацій тощо. На основі цього еталонного профілю здійснено порівняння з характеристиками шести гнучких методологій. Для кожної методології обчислено відстань Хеммінга, що демонструє кількість невідповідностей між потребами проєкту та можливостями підходу. Чим менше значення, тим вища потенційна відповідність методології для реалізації продуктового проєкту.

У таблиці наведено результати цього порівняння (таблиця 3.1).

Таблиця 3.1 - Матриця відповідності Agile-методологій критеріям продуктового проєкту

Критерій	Scrum	Kanban	XP	Lean	Scrumban	Crystal
Визначеність вимог	1	1	1	1	1	0
Рівень залучення замовника	1	0	1	0	1	1
Масштаб проєкту	1	1	1	1	1	1
Тривалість виконання	1	1	1	1	1	1
Критичність помилок	1	0	1	0	0	1
Інноваційність задачі	0	1	1	1	1	0
Потреба в інтеграції	0	0	1	1	0	1
Регуляторні вимоги	1	1	1	1	1	0
Зрілість команди	1	0	0	0	1	1
Частота змін вимог	1	1	1	1	1	0

Отже, порахуємо відстань Хеммінга між ідеальним вектором критеріїв для продуктового проєкту та відповідними векторами кожної з гнучких методологій (таблиця 3.2).

Таблиця 3.2 - Векторна модель і відстань Хеммінга між гнучкими методологіями

Гнучкі методології	Вектор	Відстань Хеммінга
Crystal	[0, 1, 1, 1, 1, 0, 1, 0, 1, 0]	2
XP	[1, 1, 1, 1, 1, 1, 1, 1, 0, 1]	3
Scrum	[1, 1, 1, 1, 1, 0, 0, 1, 1, 1]	4
Scrumban	[1, 1, 1, 1, 0, 1, 0, 1, 1, 1]	4
Lean	[1, 0, 1, 1, 0, 1, 1, 1, 0, 1]	5
Kanban	[1, 0, 1, 1, 0, 1, 0, 1, 0, 1]	6

Підсумкова таблиця відстаней Хеммінга для продуктового проєкту показує, що деякі методології значно краще відповідають ключовим вимогам: гнучкості, інноваційності, масштабованості та активній взаємодії із замовником. Найменша відстань свідчить про високу релевантність моделі до потреб продуктової розробки. Натомість моделі з меншою адаптивністю поступаються через недостатню підтримку змін і складних бізнес-вимог.

2.3.2 Аутсорс проєкти

Аутсорсингові ІТ-проєкти зазвичай реалізуються для зовнішніх клієнтів і мають чіткі договірні зобов’язання. Їх вирізняє визначеність вимог, формалізована комунікація, необхідність дотримання термінів і стандартів, а також висока відповідальність за якість продукту. У таких проєктах часто важливо забезпечити масштабування, надійність та дотримання контрактних умов.

Ідеальний вектор вимог для аутсорсингових проєктів такий:

[1, 0, 1, 1, 1, 0, 1, 1, 1, 0].

У таблиці представлено оцінку здатності кожної з гнучких методологій задовольняти ці критерії. 1 позначає наявність підтримки, 0 - її відсутність (таблиця 3.3).

Таблиця 3.3 - Матриця відповідності Agile-методологій критеріям аутсорс проєкту

Критерій	Scrum	Kanban	XP	Lean	Scrumban	Crystal
1	2	3	4	5	6	7
Визначеність вимог	0	0	0	0	0	1
Рівень залучення замовника	1	0	1	0	1	1

Кінець таблиці 3.3

1	2	3	4	5	6	7
Масштаб проєкту	1	1	1	1	1	1
Тривалість виконання	0	0	0	0	0	0
Критичність помилок	0	1	0	1	1	0
Інноваційність задачі	1	0	0	0	0	1
Потреба в інтеграції	1	1	0	0	0	1
Регуляторні вимоги	0	0	0	0	0	1
Зрілість команди	1	0	0	0	1	1
Частота змін вимог	0	0	0	0	0	1

Отже, порахуємо відстань Хеммінга між ідеальним вектором критеріїв для аутсорс проєкту та відповідними векторами кожної з гнучких методологій (таблиця 3.4).

Таблиця 3.4 - Векторна модель і відстань Хеммінга між гнучкими методологіями

Гнучкі методології	Вектор	Відстань Хеммінга
Kanban	[0, 0, 1, 0, 1, 0, 1, 0, 0, 0]	4
Crystal	[1, 1, 1, 0, 0, 1, 1, 1, 1, 1]	5
Lean	[0, 0, 1, 0, 1, 0, 0, 0, 0, 0]	5
Scrumban	[0, 1, 1, 0, 1, 0, 0, 0, 1, 0]	5
Scrum	[0, 1, 1, 0, 0, 1, 1, 0, 1, 0]	6
XP	[0, 1, 1, 0, 0, 0, 0, 0, 0, 0]	7

Для аутсорсингових ІТ-проєктів важливими є чітко визначені вимоги, масштабованість, інтеграція з іншими системами та досвід команди. Результати підрахунку відстаней Хеммінга показали, що деякі моделі мають вищу відповідність за рахунок гнучкої структури, що дозволяє адаптуватися

до зовнішніх залежностей і бізнес-вимог. Методології з більшою кількістю відмінностей, навпаки, можуть не забезпечити достатньої підтримки специфіки аутсорсингового підходу.

2.3.3 R&D проєкти

Дослідницько-орієнтовані проєкти (R&D) зосереджені на створенні інновацій, тестуванні нових ідей, розробці прототипів і дослідженні технологій. Їм притаманні висока невизначеність, мінлива специфікація, а також потреба в експериментах. Часто відсутня необхідність дотримання формальних вимог або регуляцій.

Ідеальний вектор потреб R&D-проєктів виглядає наступним чином:

[0, 1, 1, 0, 0, 1, 0, 0, 1, 1].

У таблиці зазначено, наскільки кожна модель управління здатна відповідати цим критеріям. 1 означає відповідність, 0 - невідповідність (таблиця 3.5).

Таблиця 3.5 - Матриця відповідності Agile-методологій критеріям R&D проєкту

Критерій	Scrum	Kanban	XP	Lean	Scrumban	Crystal
1	2	3	4	5	6	7
Визначеність вимог	1	1	1	1	1	0
Рівень залучення замовника	0	1	0	1	0	0
Масштаб проєкту	1	1	1	1	1	1
Тривалість виконання	1	1	1	1	1	1
Критичність помилок	1	0	1	0	0	1

Кінець таблиці 3.5

1	2	3	4	5	6	7
Інноваційність задачі	0	1	1	1	1	0
Потреба в інтеграції	0	0	1	1	0	1
Регуляторні вимоги	1	1	1	1	1	0
Зрілість команди	0	1	1	1	0	0
Частота змін вимог	1	1	1	1	1	0

Отже, порахуємо відстань Хеммінга між ідеальним вектором критеріїв для R&D проєкту та відповідними векторами кожної з гнучких методологій (таблиця 3.6).

Таблиця 3.6 - Векторна модель і відстань Хеммінга між гнучкими методологіями

Гнучкі методології	Вектор	Відстань Хеммінга
Kanban	[1, 1, 1, 1, 0, 1, 0, 1, 1, 1]	3
Lean	[1, 1, 1, 1, 0, 1, 1, 1, 1, 1]	4
Scrumban	[1, 0, 1, 1, 0, 1, 0, 1, 0, 1]	5
XP	[1, 0, 1, 1, 1, 1, 1, 1, 1, 1]	6
Scrum	[1, 0, 1, 1, 1, 0, 0, 1, 0, 1]	7
Crystal	[0, 0, 1, 1, 1, 0, 1, 0, 0, 0]	7

Підсумкова таблиця відстаней Хеммінга для R&D проєкту показує, що деякі підходи значно краще відповідають ключовим вимогам: інноваційності, гнучкості та здатності швидко адаптуватися до змін. Найменша відстань свідчить про високу релевантність моделі до особливостей науково-дослідницької діяльності. Натомість підходи з меншою адаптивністю поступаються через обмежену підтримку експериментів та складних бізнес-вимог.

2.3.4 Проєкти технічної підтримки

Проєкти технічної підтримки мають на меті підтримання стабільності систем, усунення інцидентів і своєчасну реакцію на запити. У них особливо важливими є передбачуваність, стабільність, відповідність нормативам та ефективне управління завданнями. Інноваційність або гнучка зміна вимог зазвичай не є пріоритетом.

Ідеальний вектор виглядає так:

[1, 0, 1, 1, 1, 0, 1, 1, 1, 0].

У таблиці відображено, наскільки ефективно гнучкі методології покривають ці потреби. 1 - критерій підтримується, 0 - не підтримується (таблиця 3.7).

Таблиця 3.7 - Матриця відповідності Agile-методологій критеріям проєкту технічної підтримки

Критерій	Scrum	Kanban	XP	Lean	Scrumban	Crystal
Визначеність вимог	0	0	0	0	0	1
Рівень залучення замовника	0	1	0	1	0	0
Масштаб проєкту	1	1	1	1	1	1
Тривалість виконання	0	0	0	0	0	0
Критичність помилок	0	1	0	1	1	0
Інноваційність задачі	0	1	1	1	1	0
Потреба в інтеграції	1	1	0	0	1	0
Регуляторні вимоги	0	0	0	0	0	1
Зрілість команди	0	1	1	1	0	0
Частота змін вимог	0	0	0	0	0	1

Отже, порахуємо відстань Хеммінга між ідеальним вектором критеріїв для проєкту технічної підтримки та відповідними векторами кожної з гнучких методологій (таблиця 3.8).

Таблиця 3.8 - Векторна модель і відстань Хеммінга між гнучкими методологіями

Гнучкі методології	Вектор	Відстань Хеммінга
Kanban	[0, 1, 1, 0, 1, 1, 1, 0, 1, 0]	5
Scrumban	[0, 0, 1, 0, 1, 1, 1, 0, 0, 0]	5
Crystal	[1, 0, 1, 0, 0, 0, 0, 1, 0, 1]	5
Scrum	[0, 0, 1, 0, 0, 0, 1, 0, 0, 0]	5
XP	[0, 0, 1, 0, 0, 1, 0, 0, 1, 0]	6
Lean	[0, 1, 1, 0, 1, 1, 0, 0, 1, 0]	6

Підсумкова таблиця відстаней Хеммінга для проєкту підтримки показує, що деякі моделі краще адаптовані до стабільного середовища з чітко визначеними вимогами, обмеженим рівнем змін та високою критичністю помилок. Найменша відстань у таблиці свідчить про високу відповідність таким ключовим аспектам, як передбачуваність, підтримка довготривалих циклів розробки та інтеграція з існуючими системами. Натомість менш релевантні підходи демонструють слабшу придатність до умов, де важливою є стабільність, чітка регламентація процесів і низька частота змін.

2.3.5 Внутрішньо корпоративні проєкти

Внутрішні корпоративні проєкти реалізуються для покращення внутрішніх процесів компанії, інтеграції систем або забезпечення відповідності внутрішнім стандартам. Їм властиве поєднання високих вимог

до інтеграції, регуляторної відповідності, технічної складності та необхідності взаємодії з багатьма зацікавленими сторонами.

Ідеальний вектор виглядає наступним чином:

[1, 1, 1, 1, 1, 0, 1, 1, 1, 0].

У таблиці подано, наскільки методології відповідають цим умовам. 1 свідчить про наявність відповідної підтримки, 0 - про її відсутність (таблиця 3.9).

Таблиця 3.9 - Матриця відповідності Agile-методологій критеріям внутрішнього проєкту

Критерій	Scrum	Kanban	XP	Lean	Scrumban	Crystal
Визначеність вимог	0	0	0	0	0	1
Рівень залучення замовника	1	0	1	0	1	1
Масштаб проєкту	1	1	1	1	1	1
Тривалість виконання	1	1	1	1	1	1
Критичність помилок	1	0	1	0	0	1
Інноваційність задачі	1	0	0	0	0	1
Потреба в інтеграції	1	1	0	0	1	0
Регуляторні вимоги	0	0	0	0	0	1
Зрілість команди	1	0	0	0	1	1
Частота змін вимог	0	0	0	0	0	1

Отже, порахуємо відстань Хеммінга між ідеальним вектором критеріїв для внутрішнього корпоративного проєкту та відповідними векторами кожної з гнучких методологій (таблиця 3.10).

Таблиця 3.10 - Векторна модель і відстань Хеммінга між гнучкими методологіями

Гнучкі методології	Вектор	Відстань Хеммінга
Crystal	[1, 1, 1, 1, 1, 1, 0, 1, 1, 1]	3
Scrumban	[0, 1, 1, 1, 0, 0, 1, 0, 1, 0]	3
Scrum	[0, 1, 1, 1, 1, 1, 1, 0, 1, 0]	3
XP	[0, 1, 1, 1, 1, 0, 0, 0, 0, 0]	4
Kanban	[0, 0, 1, 1, 0, 0, 1, 0, 0, 0]	5
Lean	[0, 0, 1, 1, 0, 0, 0, 0, 0, 0]	6

Підсумкова таблиця відстаней Хеммінга для внутрішнього проєкту показує, що деякі підходи значно краще відповідають ключовим вимогам: передбачуваності, збалансованого планування, відповідності внутрішнім регламентам і підтримки стабільної взаємодії між командами. Найменші відстані вказують на високу релевантність обраних моделей до умов внутрішньої розробки, де важливу роль відіграють контрольоване середовище, чітка структура і взаємодія з іншими підрозділами організації. Підходи з більшою відстанню поступаються через недостатню підтримку внутрішніх процесів та вимог корпоративного рівня.

2.4 Висновики з другого роздилу

Розроблена система критеріїв оцінювання відповідності гнучких гібридних моделей життєвого циклу до різновидів ІТ-проєктів має вагоме практичне значення як у стратегічному плануванні, так і в оперативному управлінні. У середовищі, де ІТ-проєкти суттєво різняться за своїми вимогами, тривалістю, інноваційністю та масштабом, відсутність уніфікованого підходу до вибору методології часто призводить до зниження продуктивності команд,

зростання витрат, втрат у якості та непередбачуваності результатів.

Використання системи з чітко визначеними критеріями дозволяє перейти від інтуїтивного вибору до формалізованого, доказового підходу, який базується на порівнянні бінарних векторів за допомогою відстані Хеммінга. Такий підхід є:

− масштабованим - може бути застосований як до окремих команд, так і до програм управління проєктним портфелем у великих організаціях;

− універсальним - дає змогу адаптувати критерії до специфіки організації або галузі;

− орієнтованим на якість - мінімізує ризики, пов’язані з неправильним вибором методології на старті проєкту;

− прозорим - забезпечує можливість аудитування управлінських рішень;

− гнучким - дає змогу враховувати вагу кожного критерію залежно від пріоритетів бізнесу чи організаційної політики.

Проведений аналіз відповідності методологій потребам п’яти ключових типів IT-проєктів (продуктових, аутсорсингових, R&D, підтримки, внутрішніх корпоративних) показав, що немає жодної ідеально універсальної методології, яка б безкомпромісно задовольняла всі можливі варіації вимог. Однак за сукупністю факторів методологія Crystal виявилася найбільш релевантною до більшості типів проєктів, оскільки:

− налаштовується під масштаб проєкту (варіанти Clear, Orange, Red);

− гнучка та адаптивна, що дозволяє пристосовуватися до конкретного контексту;

− підтримує мінімальну бюрократію та місцеву оптимізацію - це особливо цінно в умовах динамічного середовища;

− забезпечує індивідуальний підхід до команд, що позитивно позначається на їх мотивації та продуктивності;

− успішно застосовується як у продуктових, так і у внутрішніх корпоративних та аутсорсингових ініціативах.

Хоча для кожного окремого випадку варто проводити індивідуальну оцінку, Crystal можна обрати як базову методологію, що найменше потребує адаптації в межах усіх розглянутих типів IT-проєктів. Вона здатна виступати каркасом, до якого за потреби додаються практики інших фреймворків (наприклад, Kanban-дошки для візуалізації, XP-практики для технічного вдосконалення, Scrum-ітерації для регулярності).

Таким чином, вибір Crystal як уніфікованого підходу з урахуванням сформованої системи критеріїв дозволяє забезпечити баланс між структурністю й адаптивністю, мінімізувати ризики невідповідності, покращити командну ефективність і забезпечити масштабованість управлінських рішень в умовах проєктної диверсифікації.

3 РОЗРОБКА РЕКОМЕНДАЦІЙ З АДАПТАЦІЇ ГІБРИДНОЇ МОДЕЛІ CRYSTAL ДО ОСОБЛИВОСТЕЙ ОКРЕМИХ РАЗНОВИДІВ ІТ-ПРОЄКТІВ

3.1 Розробка рекомендацій з адаптації моделі Crystal до особливостей продуктових ІТ проєктів

Продуктові ІТ-проєкти є однією з найскладніших форм проєктної діяльності в галузі інформаційних технологій. Вони спрямовані на створення інноваційного цифрового продукту, орієнтованого на кінцевого користувача, і, як правило, супроводжуються високим ступенем невизначеності, швидкими темпами змін, необхідністю оперативної реакції на зворотний зв’язок з ринку та значним технічним боргом, що виникає через інтенсивний темп ітерацій. У зв’язку з цим управління такими проєктами вимагає не лише гнучкості, але й адаптивності до змін вимог, високої залученості команди та постійного вдосконалення процесів розробки.

Методологія Crystal, запропонована Алістером Коберном, є сімейством гнучких підходів до розробки програмного забезпечення, побудованих на принципах мінімізації бюрократії, посилення міжособистісної взаємодії, постійного покращення і адаптації процесів до контексту проєкту. Crystal не є жорстким фреймворком із фіксованим набором артефактів та ритуалів. Навпаки, її суть полягає в побудові процесу «знизу вгору», виходячи з фактичних потреб проєкту, розміру команди, критичності помилок, комунікаційної інфраструктури та інших факторів.

У ході проведеного дослідження, що базується на системі бінарних критеріїв та обчисленні відстані Хеммінга між вимогами до конкретного типу ІТ-проєкту та характеристиками гнучких методологій, було встановлено, що Crystal має найвищий рівень відповідності продуктовим проєктам, отримавши мінімальну відстань 2 (з 10 можливих розбіжностей). Це свідчить про те, що Crystal якнайкраще охоплює типові потреби продуктової розробки: гнучкість

до змін, підтримку складної архітектури, тривалий життєвий цикл, високу інтенсивність взаємодії з користувачем та значну інноваційну складову.

У зв’язку з цим було розроблено комплекс практичних рекомендацій щодо адаптації Crystal до особливостей продуктових проєктів:

1. Вибір відповідного підтипу методології.

Зважаючи на масштаб продуктової команди, слід обирати між різними варіантами Crystal. Для невеликих команд до 6-8 осіб (наприклад, стартапів або незалежних продуктових груп у великих компаніях) доцільним є використання Crystal Clear. Він забезпечує високу швидкість адаптації, мінімальний набір обов’язкових ролей і документів. Для більших команд (10-40 учасників), що розробляють масштабні SaaS-рішення чи багатокомпонентні системи, рекомендовано застосування Crystal Orange, який вводить додаткові інструменти комунікації, координації та тестування.

2. Інкорпорація зворотного зв’язку користувачів.

Продуктові команди повинні активно інтегрувати практики роботи з користувацькими відгуками в цикл розробки. В межах Crystal це можна реалізувати через встановлення регулярних інтервалів валідації функціоналу, використання A/B-тестування, інтерв’ю з користувачами, UX-дослідження та метрики залучення. Ці заходи доцільно поєднувати з мінімальним релізним циклом (наприклад, кожні 1-2 тижні), що відповідає принципу раннього постачання (early delivery).

3. Впровадження практик технічної дисципліни.

Хоча Crystal не диктує технічні практики, продуктові проєкти вимагають стабільності коду та високої швидкості постачання. Тому рекомендовано включати практики з Extreme Programming (XP): парне програмування, розробку через тестування (TDD), рефакторинг, автоматизацію збірки та CI/CD. Це дозволяє зменшити технічний борг, уникати збоїв при частих релізах і підвищити якість продукту.

4. Використання візуалізації та беклогів.

Хоча Crystal не вимагає ведення беку логів, у продуктових командах

важливо мати централізовану систему пріоритетів. Доцільно впровадити Product Backlog із приоритизацією за цінністю для користувача, а також Kanban-дошку або Scrum-подібний workflow для візуалізації прогресу. Це підвищує прозорість і синхронізує очікування між технічною та продуктовою частинами команди.

5. Адаптація командних ролей.

У Crystal офіційні ролі не є обов'язковими, однак у продуктовому контексті доцільно призначити Product Owner'а, який буде мостом між командою та ринком. Також варто виокремити технічного лідера, відповідального за архітектуру, технічні борги та узгодженість модулів. Це дозволяє зберігати стратегічний фокус і уникати фрагментації рішень у багатокомандних продуктах.

6. Підтримка інформаційної прозорості та самоорганізації.

Crystal сприяє побудові довіри та самостійності команд, однак для продуктових ініціатив із високими бізнес-ставками важливо зберігати ритм та передбачуваність. Рекомендується впровадження щотижневих стендапів, щомісячних оглядів прогресу, ретроспектив та внутрішньої документації (наприклад, Confluence). Така практика дозволяє уникати втрати контексту при масштабуванні команди або зміні складу.

7. Гнучка інтеграція з іншими системами.

Продуктові IT-проєкти часто вимагають масштабної інтеграції з аналітичними платформами, платіжними системами, сторонніми API. У Crystal варто виокремити інтеграційні завдання як окремий тип активності з фокусом на стабільність, тестування, відповідність SLA та автоматизовану перевірку змін, що допоможе підтримувати якість взаємодії з зовнішніми сервісами.

Таким чином, запропоновані рекомендації орієнтовані на поєднання адаптивної природи Crystal із специфікою продуктової розробки, яка вимагає балансування між швидкістю інновацій, стабільністю та якістю. Використання Crystal у продуктових командах дозволяє зберігати гнучкість управління,

уникаючи надмірної регламентації, і водночас забезпечує рамки, необхідні для ефективної командної взаємодії, масштабування і відповідності ринковим вимогам. При цьому методологія залишається відкритою до подальшої еволюції відповідно до змін у середовищі, що є однією з головних її переваг у контексті продуктових проєктів.

3.2 Розробка рекомендацій з адаптації моделі Crystal до особливостей аутсорсингових IT проєктів

Аутсорсингові IT-проєкти мають специфічну природу, відмінну від продуктових або внутрішніх ініціатив. Основною їх особливістю є те, що вони реалізуються на замовлення сторонніх організацій, часто з інших галузей, для яких розробка програмного забезпечення не є основним видом діяльності. У таких проєктах акцент робиться не стільки на інноваціях чи гнучкому функціоналі, скільки на точному виконанні вимог, дотриманні термінів, бюджету, технічної специфікації та формалізованих контрактних зобов’язань.

Хоча традиційно аутсорсинг асоціюється з каскадними підходами (Waterfall), зростаюча складність замовлень і попит на швидку реалізацію рішень змушують компанії впроваджувати елементи гнучкості. Методологія Crystal, що відрізняється адаптивністю, орієнтацією на контекст і мінімізацією бюрократії, може бути ефективною основою управління аутсорсинговими проєктами - за умови її відповідного налаштування.

На основі зіставлення системи критеріїв та результатів відстані Хеммінга, було визначено, що Crystal посідає друге місце за рівнем відповідності до аутсорсингових IT-проєктів, демонструючи відносну гнучкість, але потребуючи адаптації до низки критичних аспектів, таких як: чіткість вимог, формалізація процесів, регулярна звітність та обмежена автономія команд.

З урахуванням зазначеного, розроблено такі рекомендації щодо адаптації моделі Crystal для використання в аутсорсинговому контексті:

1. Підвищення рівня формалізації вимог і документації.

На відміну від продуктових команд, в аутсорсингових проєктах вимоги, технічна специфікація і критерії прийняття результатів часто фіксуються в контракті.

Відповідно, Crystal має бути доповнена процесами документування, що включають докладні технічні специфікації, Acceptance Criteria для кожної функції, а також контрольні списки для демонстрацій і релізів.

Ці документи мають бути обов’язковими артефактами, які перевіряються на кожному етапі проєкту. Вони також повинні підтверджуватися замовником.

2. Запровадження регулярного зовнішнього звітування.

Оскільки в аутсорсингових проєктах замовник часто перебуває поза командою, критично важливо забезпечити регулярну прозору звітність.

До Crystal слід інтегрувати щотижневі статус-репорти, онлайн-дашборди прогресу (наприклад, через Jira або Azure DevOps), узгоджені KPI, а також заплановані онлайн-сесії з демонстрацією прогресу, такі як sprint demo або milestone review.

Це дозволяє підтримувати довіру клієнта. Завдяки цьому зменшуються ризики непорозумінь і запобігається "розрив очікувань".

3. Посилення контролю версій і якості.

На відміну від гнучких продуктових команд, які можуть релізити інкременти за власним графіком, в аутсорсингу релізи часто пов’язані з контрактними зобов’язаннями.

Тому слід впровадити обов’язкові етапи тестування (unit, integration, acceptance), практики code review та контроль артефактів, систему версіювання з управлінням релізами (semantic versioning), а також протокол передрелізного узгодження (release readiness checklist).

Це гарантує відповідність результатів очікуванням замовника. Також

такі заходи полегшують технічну підтримку.

4. Визначення формальних ролей.

Незважаючи на декларативну гнучкість Crystal щодо ролей, для аутсорсингових проєктів важливо мати чітку структуру відповідальності.

До ключових належать Project Manager як основний контакт для замовника та координатор процесів, Team Lead або Technical Lead, відповідальний за технічне виконання і архітектурну цілісність, Business Analyst, що координує вимоги між замовником і командою, та QA Lead, який контролює якість і тестування.

Ці ролі мають бути явно визначені в проєктному плані. Вони також повинні бути закріплені в комунікаційній схемі.

5. Використання Scrumban-елементів для візуалізації та планування.

Хоча Crystal дозволяє варіювати інструменти планування, для забезпечення контрольованості у проєктах з фіксованими термінами доцільно поєднати візуальні Kanban-дошки з обмеженням WIP, Timeboxing, наприклад, двотижневі спринти, а також ретроспективи для командного вдосконалення без порушення контрактних зобов’язань.

Такий підхід дає змогу зберігати гнучкість процесів без втрати керованості.

6. Адаптація до культурних і часових бар'єрів.

Багато аутсорсингових проєктів реалізуються в умовах розподіленої команди, часто з замовником в іншій часовій зоні.

У межах Crystal рекомендовано встановити стандартизовані години синхронної комунікації, використовувати централізовану документацію, наприклад через Confluence, впровадити культуру письмових протоколів після зустрічей та надавати доступ до записів демонстрацій, таких як відео, презентації чи тестові кейси.

Ці заходи допомагають мінімізувати ризики втрати контексту. Вони також підвищують якість взаємодії з клієнтом.

Таким чином, методологія Crystal може бути ефективно адаптована до

умов аутсорсингової розробки, якщо буде доповнена необхідними елементами формалізації, звітності, контролю та структурованої взаємодії. Попри її орієнтацію на гнучкість, Crystal дає змогу створити легку, адаптивну рамку, в якій можуть співіснувати вимоги контрактного проєкту і потреба в командній автономії.

Запропоновані рекомендації дозволяють зберегти гнучкість усередині команди, одночасно забезпечуючи відповідність зовнішнім очікуванням, що є ключовим фактором успіху аутсорсингових ініціатив. У результаті, Crystal може стати основою побудови індивідуалізованих процесів розробки у компаніях, що працюють з широким колом клієнтів і прагнуть зберігати конкурентоспроможність за рахунок якості, прозорості та адаптивності.

3.3 Розробка рекомендацій з адаптації моделі Crystal до особливостей R&D IT проєктів

Дослідницько-орієнтовані проєкти у сфері інформаційних технологій (R&D - Research and Development) мають низку унікальних характеристик, що істотно відрізняють їх від продуктових або аутсорсингових ініціатив. Основною метою таких проєктів є не створення готового програмного продукту, а розробка інноваційних технологій, створення прототипів, генерація нових знань або перевірка гіпотез. У цьому контексті критично важливими є гнучкість процесів, допуск до високого рівня невизначеності, толерантність до помилок, можливість експериментування, а також мінімізація формальних обмежень.

Методологія Crystal, завдяки своїй високій адаптивності, персоналізованості під конкретний проєкт та мінімальному рівню бюрократії, є природним кандидатом для застосування в R&D-проєктах. Проте, враховуючи специфіку дослідницького процесу, виникає потреба у

розширенні базових принципів Crystal з урахуванням науково-дослідного контексту, а також інтеграції з технічними та управлінськими підходами, що підтримують експериментальну діяльність

На основі порівняльного аналізу критеріїв відповідності та результатів обчислення відстані Хеммінга, встановлено, що Crystal демонструє помірний рівень відповідності до вимог R&D-проєктів. Для підвищення ефективності її застосування доцільно врахувати низку адаптаційних рекомендацій:

1. Опора на короткі, експериментальні ітерації без фіксованого очікуваного результату.

У дослідницьких проєктах ключове значення має швидке тестування гіпотез, верифікація технічної здійсненності та оцінка практичної доцільності ідей. У межах Crystal доцільно впровадити короткі ітерації (1-2 тижні), які завершуються прототипом, proof of concept (PoC), технічним висновком або невдалим експериментом, що документується. Важливо, щоб метою ітерації було не досягнення завершеного функціоналу, а здобуття нових знань, що впливають на подальші рішення.

2. Формування мультидисциплінарних команд з фокусом на знання, а не на ролі.

У R&D-проєктах ключову роль відіграє науково-технічна експертиза. Тому Crystal необхідно адаптувати шляхом створення команд із гнучким розподілом ролей, де фокус зміщується з формальної структури на компетенції. Основу команди повинні складати фахівці, здатні до самоорганізації та швидкого обміну знаннями: інженери-дослідники, архітектори, аналітики, дата-сайентисти. У такому контексті не обов’язково призначати Product Owner’а або Project Manager’а у класичному розумінні - їх функції можуть виконувати провідні спеціалісти, що володіють баченням дослідницької мети.

3. Інтеграція з техніками дослідницького проєктування (Design Thinking, Lean R&D).

Методологія Crystal допускає використання зовнішніх практик. У

дослідницьких ІТ-проєктах доцільно інтегрувати елементи Design Thinking (емпатія, формулювання проблеми, генерація рішень, прототипування, тестування) та Lean-підходів, зокрема визначення метрик успіху експерименту, валідації результатів з мінімальними витратами та ітеративного відбору напрямків. Це дозволяє зменшити витрати на неперспективні гіпотези й підвищити ефективність використання ресурсів.

4. Пріоритетність документації знань замість технічної специфікації.

Оскільки вимоги до результатів R&D-проєктів часто змінюються, класичне документування специфікацій стає неефективним.

Натомість у Crystal для R&D доцільно впровадити ведення "журналу досліджень" (research log) - документа, який містить формулювання гіпотез, методи перевірки, очікувані та фактичні результати, а також причини зміни напряму.

Така форма дозволяє зберігати контекст дослідження. Вона також сприяє наступності знань між ітераціями та командами.

5. Відмва від жорсткого планування та фіксованих дедлайнів на користь гнучкого роадмапу.

На етапі дослідження складно передбачити конкретний обсяг завдань чи терміни реалізації. Тому Crystal слід адаптувати, відмовившись від фіксованого календарного планування на користь динамічного роадмапу з віхами («milestones»), які описують не функціонал, а цілі експерименту або рівень досягнення зрозумілості проблеми. Кожна віха повинна мати чітко описану мету, критерії успіху та допустимі ризики.

6. Фокус на внутрішній мотивації та науковій автономії команди.

У R&D-проєктах стандартні інструменти зовнішнього контролю (KPI, velocity, burndown-діаграми) мають низьку ефективність. Натомість рекомендовано створити середовище, яке підтримує інтелектуальну автономію, дослідницький інтерес та свободу у виборі підходів. Crystal, як методологія, що від початку передбачає адаптацію під команду, має стати базою для створення гнучкого, але неформалізованого середовища, де

основною метою є досягнення прориву, а не дотримання процесу.

Таким чином, методологія Crystal може бути органічно адаптована до специфіки R&D IT-проєктів, оскільки відповідає ключовим характеристикам дослідницької діяльності: високій динамічності, потребі у гнучкості, відсутності стабільних вимог і акценту на міждисциплінарній командній роботі. Проте її ефективне застосування потребує свідомого розширення базового інструментарію Crystal і його доповнення практиками з галузей інноваційного дизайну, експериментального аналізу та Lean-досліджень.

Запропоновані рекомендації дозволяють сформувати на основі Crystal таку організацію проєктного процесу, яка буде не лише адаптивною до змін, а й стимулюватиме команду до самостійного пошуку нових ідей, швидкої перевірки гіпотез і генерації знань - основного ресурсу R&D-проєктів. Таким чином, Crystal може стати ефективною методологічною основою для управління інноваційними IT-ініціативами в академічному, корпоративному або стартап-середовищі.

3.4 Розробка рекомендацій з адаптації моделі Crystal до особливостей IT проєктів технічної підтримки

IT-проєкти технічної підтримки становлять специфічний клас проєктної діяльності, метою якої є забезпечення безперервної, стабільної та надійної роботи вже впроваджених інформаційних систем. Основні завдання в межах таких проєктів включають обробку інцидентів, усунення помилок, впровадження змін на запит, моніторинг продуктивності, оновлення систем та забезпечення відповідності вимогам SLA (Service Level Agreement). У цьому контексті пріоритетами є оперативність реагування, передбачуваність, відтворюваність дій, а також мінімізація ризиків змін.

На відміну від R&D або продуктових ініціатив, де високий рівень невизначеності та інноваційності є бажаним або неминучим, у проєктах підтримки навпаки важлива стабільність, мінімізація змін і технічна акуратність. Методологія Crystal, попри свій первинний фокус на гнучкість і адаптивність, може бути результативно використана і в середовищі технічної підтримки, якщо вона буде адаптована відповідно до його специфіки.

Аналіз відповідності гнучких методологій до потреб проєктів технічної підтримки, здійснений шляхом обчислення відстані Хеммінга, показав, що Crystal має потенціал до застосування в цій сфері, але потребує уточнення акцентів і структурних змін у практичному впровадженні. З огляду на це, сформульовано наступні рекомендації:

1. Організація Crystal-команди як сервісної структури з розмежуванням рівнів підтримки.

У рамках технічної підтримки доцільно організовувати команду за принципами L1/L2/L3 - першого, другого та третього рівнів реагування. Crystal, у цьому випадку, має бути адаптована до моделі, де кожна підгрупа команди має власні зони відповідальності, SLA-орієнтовану мету та пріоритетність задач. Це дозволяє зберігати гнучкість комунікації й

автономність, властиві Crystal, водночас забезпечуючи структуровану відповідь на запити користувачів.

2. Формалізація каналів надходження задач та реєстрації інцидентів

Для уникнення хаотичної обробки звернень необхідно чітко регламентувати процеси.

Сюди входять формати створення задач (через Service Desk, Jira або інші системи управління заявками), класифікація запитів (інцидент, запит на зміну, проблема, покращення) та механізм реєстрації історії звернень для подальшого аналізу.

У Crystal ці процеси слід закріпити як адаптовані правила взаємодії з зовнішнім світом, що є частиною проєктної політики.

3. Впровадження практик безперервного покращення без ризику для продуктивного середовища.

Хоча Crystal не передбачає жорсткої фіксації регламентів, доцільним є використання внутрішніх мікро-ретроспектив - коротких зустрічей команди підтримки для виявлення повторюваних помилок, оптимізації рутинних процесів, впровадження автоматизації. Важливо, щоб результати таких покращень проходили через контроль якості й були реалізовані поступово, із тестуванням у середовищі, відмінному від продуктивного.

4. Встановлення метрик і моніторингу відповідно до SLA.

Оскільки підтримка працює в умовах обмежень часу реакції, відновлення та вирішення проблем, Crystal має бути доповнена системою обліку метрик.

Вона має фокусуватися на часі реагування та часу вирішення (Response Time, Resolution Time), частоті повторних інцидентів, відсотку звернень, що потребували ескалації, а також частці автоматизованих відповідей.

Ці метрики не повинні обмежувати гнучкість, але служать інструментом самооцінки ефективності команди.

5. Застосування Kanban-дошки як основного інструмента візуалізації задач.

Для підтримки прозорості процесів у режимі безперервного потоку завдань найбільш ефективним є використання Kanban-дошки в адаптації до Crystal.

Рекомендовано впровадити колонки на кшталт "Нові", "У роботі", "Очікує відповіді", "Готово", "Відхилено", обмежити кількість задач у кожній з них (WIP-ліміти) та сортувати задачі за пріоритетами (P1-P4), що відповідають рівню критичності.

Така організація дозволяє команді оперативно перемикатися між завданнями, не втрачаючи цілісності процесу.

6. Чітке визначення обмежень і рамок відповідальності команди підтримки.

Методологія Crystal передбачає гнучкий розподіл обов’язків, проте у підтримці важливо формалізувати зони впливу команди, особливо коли йдеться про системи, які обслуговуються зовнішніми постачальниками, або коли втручання в код можливе лише через окремі канали.

Рекомендовано задокументувати, що саме входить у зону відповідальності команди (сервіси, інтерфейси, модулі), які типи змін вона може вносити самостійно, а також у яких випадках потрібна ескалація чи залучення третьої сторони.

Такий підхід дозволяє зберігати чіткість процесів у межах гнучкої методології.

Таким чином, методологія Crystal після відповідної адаптації може бути ефективно застосована в контексті технічної підтримки. Вона дозволяє зберегти гнучкість та автономність команди, водночас забезпечуючи відповідність жорстким регламентам, операційним метрикам і SLA, які є невід’ємною складовою таких проєктів. Особливу цінність Crystal має у підтримці середовищ із широким спектром компонентів, частими дрібними змінами, вимогами до високої надійності та одночасним прагненням до вдосконалення.

Застосування наведених рекомендацій сприятиме формуванню

ефективної культури підтримки, яка поєднує передбачуваність і дисципліну з командною гнучкістю, відкритістю до вдосконалення та відповідальністю за результат. У результаті, Crystal може посісти своє місце серед сучасних підходів до організації команд підтримки - особливо у тих компаніях, які прагнуть інтегрувати обслуговування систем у загальну культуру гнучкої розробки.

3.5 Розробка рекомендацій з адаптації моделі Crystal до особливостей внутрішньо корпоративних IT проєктів

Внутрішньо корпоративні IT-проєкти - це ініціативи, що реалізуються всередині організації з метою оптимізації внутрішніх процесів, автоматизації рутинних операцій, підвищення ефективності управління або забезпечення відповідності внутрішнім політикам та зовнішнім регуляторним вимогам. Відмінною рисою таких проєктів є те, що їх замовником виступає не зовнішній клієнт, а внутрішній підрозділ компанії, часто без формалізованої ролі "продуктового власника", чіткого фінансового бюджету чи оцінки комерційної доцільності.

У таких умовах типово спостерігаються: низька залученість замовника, зміщення пріоритетів під час реалізації, фрагментовані або суперечливі вимоги, значний рівень залежності від інших систем, а також ризики, пов’язані з бюрократією, політичними процесами в межах організації та проблемами інтеграції. Водночас, такі проєкти мають потенціал значного впливу на внутрішню ефективність компанії та довготривалий операційний ефект.

Методологія Crystal, завдяки своїй гнучкості, мінімальному формалізму та фокусі на адаптацію до контексту, може бути застосована для внутрішньо корпоративних ініціатив. За результатами оцінки відповідності гнучких підходів, Crystal продемонструвала найкращу відповідність цій категорії

проєктів, маючи мінімальну відстань Хеммінга - 3, що свідчить про високий рівень релевантності. Однак для ефективного впровадження потрібне цілеспрямоване налаштування під специфіку корпоративного середовища.

Нижче подано рекомендації щодо адаптації Crystal до внутрішньо корпоративних IT-проєктів:

1. Чітке формування ролі замовника або функціонального представника.

У багатьох внутрішніх проєктах відсутній виділений представник замовника, що ускладнює прийняття рішень. У межах адаптації Crystal доцільно офіційно призначити внутрішнього Product Owner'а, який представляє інтереси бізнес-підрозділу. Його обов'язки повинні включати погодження пріоритетів, валідацію вимог, прийняття результатів і контроль змін. За відсутності повноцінного PO цю функцію може виконувати аналітик або менеджер ініціативи.

2. Побудова міжфункціональної команди з урахуванням залежностей.

Через інтеграційний характер корпоративних систем, команди повинні формуватись із представників різних департаментів: розробники, тестувальники, адміністратори, бізнес-аналітики, технічні координатори тощо. Crystal дозволяє налаштовувати структуру команди під контекст, тому доцільно створити гнучке ядро, навколо якого періодично долучаються зовнішні фахівці. Особливо важливо забезпечити доступ до носіїв знань про суміжні системи (ERP, CRM, фінансові модулі).

3. Визначення фіксованих віх (milestones) із гнучким наповненням.

Внутрішньо корпоративні проєкти зазвичай мають зовнішній тиск щодо термінів (звітні періоди, регуляторні дедлайни), тому варто узгодити ключові календарні точки - віхи, пов'язані з інтеграцією, демонстраціями або затвердженням документації. Crystal підтримує гнучкість у плануванні, тому в межах кожної віхи рекомендується встановлювати гнучкий зміст робіт, що конкретизується на старті ітерації, але не обмежує команду в деталях

реалізації.

4. Формалізація політик взаємодії із суміжними системами.

Важливо враховувати, що корпоративні IT-ініціативи зазвичай мають складну архітектурну екосистему.

Тому необхідно задокументувати правила доступу до зовнішніх API, алгоритм узгодження змін із власниками сторонніх модулів, планування релізів з урахуванням технічних вікон і блокувань, а також регламент тестування в інтегрованому середовищі.

Ці політики можуть оформлюватися як додатковий додаток до командної угоди (Working Agreement) у межах Crystal.

5. Використання Scrumban-підходу для управління змінними пріоритетами.

В умовах частих змін пріоритетів, типових для внутрішніх ініціатив, доцільно реалізувати змішаний підхід.

Зокрема, ефективними є планування ітерацій зі змінним обсягом задач, візуалізація роботи на Kanban-дошці, встановлення WIP-лімітів і проведення регулярних синхронізацій із замовником.

Ця практика дозволяє зберігати контроль над потоком задач, не втрачаючи гнучкості у виконанні.

6. Оцінка ефективності через нефінансові метрики.

Оскільки внутрішні проєкти не приносять прямої комерційної вигоди, їх результативність слід оцінювати за іншими показниками.

До таких належать рівень задоволеності користувачів, час виконання змін (lead time), зменшення обсягу ручної роботи або кількості помилок, а також відповідність політикам безпеки й регуляторним вимогам.

Crystal, що не вимагає стандартизованих KPI, дозволяє адаптувати ці метрики до реальних цілей компанії.

Таким чином, внутрішньо корпоративні IT-проєкти можуть успішно реалізовуватись із використанням адаптованої версії Crystal, за умови належної структурування взаємодії з внутрішніми замовниками, формалізації

технічних залежностей і збереження гнучкості на операційному рівні. Ця методологія особливо цінна в середовищах, де одночасно необхідно поєднувати чіткість та гнучкість, а також підтримувати багатосторонню комунікацію між департаментами в межах єдиної організаційної структури.

Запропоновані рекомендації спрямовані на створення ефективної, адаптивної та довготривалої системи управління корпоративними IT-ініціативами, яка підтримує не лише впровадження рішень, а й їх сталу еволюцію в динамічному внутрішньому середовищі.

3.6 Виводи до третього розділу

На основі проведеного аналізу встановлено, що методологія Crystal завдяки своїй високій гнучкості, індивідуалізованому підходу до організації процесів та мінімалізму в регламентах може бути ефективно адаптована до широкого спектру IT-проєктів - від інноваційно-продуктових до стабілізаційних проєктів технічної підтримки.

Зокрема:

− для продуктових IT-проєктів Crystal виявився найбільш релевантним серед інших методів, адже забезпечує необхідну адаптивність до змін, інтенсивну взаємодію з користувачами та підтримку швидких ітерацій з високим рівнем технічної складності. Рекомендовано використання таких практик, як UX-інтеграція, CI/CD, рефакторинг та динамічне керування ролями;

− в аутсорсингових проєктах методологія потребує значного рівня формалізації, чіткої структури звітності, визначення ролей і фіксації вимог. Crystal може бути використана як основа, доповнена елементами контролю, контрактного менеджменту та регулярної верифікації прогресу;

− у дослідницьких R&D-проєктах Crystal демонструє природну

синергію з експериментальною природою розробки. Успішна адаптація передбачає фокус на знання, прототипування, короткі ітерації, ведення журналу досліджень та гнучке планування без фіксованих дедлайнів;

− у проєктах технічної підтримки Crystal може бути результативною за умови побудови сервісної структури з розподілом відповідальності за рівнями підтримки, впровадження SLA-метрик, чіткої класифікації звернень і Kanban-візуалізації потоку задач;

− для внутрішньо корпоративних ініціатив Crystal виявився найбільш відповідним, оскільки дає змогу будувати ефективну командну взаємодію в умовах обмеженої залученості замовника, змінних пріоритетів та великої кількості внутрішніх залежностей. Ключовими є формалізація ролі представника бізнесу, міжфункціональні команди, та орієнтація на нефінансові метрики ефективності.

Загалом, методологія Crystal може служити гнучкою основою для побудови кастомізованих під конкретний контекст процесів управління IT-проєктами. Запропоновані рекомендації демонструють її потенціал до адаптації в межах різних типів проєктів, підвищуючи ефективність розробки, зменшуючи ризики та сприяючи кращій синергії між технічними і бізнес-цілями.

4 АПРОБАЦІЯ РЕЗУЛЬТАТІВ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ

4.1 Стислий опис IT проєкту

Об’єктом розробки став вебзастосунок для прогнозу погоди, що використовує сучасні frontend- та backend-технології. Проєкт спрямований на створення зручного інструменту, який надає користувачам актуальну інформацію про погодні умови в будь-якій точці світу. Наявність такого застосунку є важливою складовою сучасного цифрового середовища, оскільки прогноз погоди впливає на повсякденне життя та планування багатьох людей.

Клієнтська частина застосунку реалізована на базі React - популярної JavaScript-бібліотеки, що дозволяє створювати інтерактивні, швидкі та адаптивні інтерфейси, зручні для користувачів на різних пристроях. React забезпечує гнучке оновлення даних на сторінці без перезавантаження, що робить роботу застосунку комфортнішою.

Серверна частина побудована на Node.js - ефективній платформі, яка дозволяє швидко обробляти запити, працювати із зовнішніми сервісами та керувати логікою бізнес-процесів. Вибір Node.js обумовлений його популярністю в розробці сучасних вебзастосунків і можливістю реалізовувати масштабовані рішення.

Основною функціональністю застосунку є надання прогнозу погоди за двома варіантами: автоматичне визначення місцезнаходження користувача за допомогою геолокації та можливість вибору міста вручну зі списку. Для отримання точних метеоданих використовується інтеграція із зовнішнім API OpenWeatherMap, що надає актуальну інформацію про температуру, вологість, швидкість і напрямок вітру, атмосферний тиск та інші параметри.

Команда розробників складалась із шести осіб: двох frontend-розробників, двох backend-розробників, UI/UX дизайнера і координатора проєкту. Кожен учасник виконував конкретні ролі, що дозволяло ефективно

розподіляти завдання та забезпечувати високу якість кінцевого продукту. UI/UX дизайнер розробляв зручний інтерфейс із урахуванням сучасних тенденцій та потреб користувачів, а координатор слідкував за плануванням, комунікацією та загальним контролем проєкту.

Спочатку для організації роботи застосовувалась методологія Kanban, що дозволяла візуалізувати статуси завдань і оперативно реагувати на зміни. Проте в процесі роботи були виявлені певні проблеми: відсутність регулярної координації між фронтендом і бекендом, неконтрольований порядок виконання задач і блокування через невидимі залежності між компонентами. Крім того, відсутність системного тестування і відкладене виявлення багів уповільнювали процес розробки.

Для забезпечення високої якості продукту команда також приділяла увагу впровадженню автоматизованого тестування і документуванню процесів. Це допомогло не тільки швидко виявляти і усувати помилки, а й підтримувати стабільність застосунку при подальшому розширенні функціоналу. Особливу увагу було приділено юзер-френдлі інтерфейсу, що відповідає сучасним стандартам зручності та доступності.

В процесі розробки також враховувалися питання безпеки, зокрема захист персональних даних користувачів і коректна робота з API. Використання сучасних підходів і протоколів дозволило знизити ризики і забезпечити стабільність і надійність роботи сервісу.

Загалом, реалізація цього проєкту стала важливим досвідом для команди, що дозволив поєднати технічні рішення з практичними підходами до організації роботи. Результатом став сучасний вебзастосунок, здатний оперативно і якісно надавати інформацію про погоду, що робить його корисним і затребуваним серед користувачів.

4.2 Результати застосування рекомендацій з адаптації обраної моделі життєвого циклу

У процесі розробки вебзастосунку для прогнозу погоди команда зіткнулася з низкою викликів, що виникали через недосконалу організацію робочого процесу. На першому етапі використовувалася методологія Kanban, яка передбачала візуалізацію завдань і контроль над їх виконанням у вигляді дошки зі статусами. Проте з часом стали очевидними її обмеження: недостатня координація між фронтендом і бекендом, непередбачувані затримки через неочевидні залежності між задачами, а також низька ефективність у роботі над задачами, що потребують тісної командної взаємодії.

З метою покращення внутрішньої комунікації, уникнення дублювання роботи, забезпечення якості та стабільності програмного забезпечення було прийнято рішення перейти на більш гнучку й адаптивну методологію - Crystal Clear. Вона розроблена спеціально для малих команд (до 6-8 осіб) і передбачає високу прозорість у комунікації, гнучкість планування, надання регулярного зворотного зв’язку й акцент на індивідуальній відповідальності.

Crystal Clear не вимагає суворого дотримання жорстких процедур, натомість передбачає адаптацію практик під конкретний проєкт і команду (таблиця 4.1). У межах цієї методології в нашій команді було впроваджено низку практик, які сприяли покращенню процесу розробки, комунікації, контролю якості та підтримки технічної документації.

Таблиця 4.1 - Реалізовані Agile-практики в межах проєкту

№	Практика	Реалізація в проєкті
1	2	3
1	Інформаційний простір	Використання Jira-дошки зі статусами: "План", "Робота", "Тест", "Готово" для прозорого відстеження завдань.

Кінець таблиці 4.1

1	2	3
2	Щотижневі демонстрації	Регулярні демо результатів ментору проєкту для зворотного зв’язку та узгодження пріоритетів.
3	Перегляд коду (Code Review)	Проведення code review двічі на тиждень, що сприяло покращенню якості коду і взаємному обміну знаннями.
4	Щоденні синхронізації	Короткі зустрічі в Google Meet для координації дій і оперативного вирішення питань.
5	Чеклисти завершення задач	Впровадження уніфікованих критеріїв "готовності" завдань для забезпечення прозорості та контролю якості.
6	Автоматизоване тестування	Впровадження unit-тестування дозволило вчасно виявляти баги і підтримувати стабільність застосунку.
7	Координація frontend і backend	Регулярні технічні обговорення і рев’ю API допомогли уникнути конфліктів та невидимих залежностей між командами.
8	Документування процесів	Підготовка та підтримка технічної документації і описів API забезпечили зрозумілість і підтримку подальшого розвитку.

Інформаційний простір став ключовим інструментом у прозорій організації роботи. Jira-дошка з чітким розподілом задач за статусами дозволила команді легко відстежувати прогрес і своєчасно реагувати на зміни. Завдяки цьому кожен учасник мав змогу бачити загальну картину роботи над проєктом, що покращувало планування й дозволяло уникати дублювання зусиль.

Щотижневі демонстрації результатів роботи перед ментором дозволяли не лише отримати оперативний зворотний зв’язок, але й регулярно оцінювати

реальний стан проєкту, уточнювати вимоги, змінювати пріоритети. Це було особливо важливо в умовах, коли частина функціоналу залежала від зовнішніх API та змін у кліматичних сервісах.

Перегляд коду став обов’язковою процедурою для всіх критичних змін у репозиторії. Двічі на тиждень відбувався взаємний code review, у межах якого розробники аналізували рішення одне одного. Це сприяло не лише підвищенню якості коду, а й зменшенню кількості помилок, формуванню єдиних підходів до написання коду, передачі знань між членами команди.

Щоденні синхронізації допомагали підтримувати динаміку роботи та уникати простоїв. Короткі зустрічі в Google Meet (до 15 хвилин) дозволяли швидко обмінюватись інформацією про поточний статус задач, узгоджувати суміжні блоки роботи та вчасно виявляти проблеми, що потребували уваги.

Особливу роль відігравали чеклисти завершення задач, які включали перелік обов’язкових кроків перед переведенням завдання у статус "Готово". Наприклад, для frontend-задач це були перевірка верстки на різних розширеннях екрана, тестування інтерактивних елементів, перевірка відповідності дизайну; для backend - перевірка на обробку помилок, коректну взаємодію з базою даних, оновлення документації.

Крім того, було впроваджено unit-тестування, що дозволяло перевіряти функціональність на рівні окремих модулів. Це суттєво знизило кількість багів при інтеграції нових компонентів і забезпечило стабільність роботи застосунку при зміні логіки або додаванні нового функціоналу.

Проблеми, що виникали через слабку інтеграцію фронтенду й бекенду, були вирішені завдяки регулярним технічним обговоренням. Перед реалізацією нових API проводилися короткі зустрічі, на яких обговорювався формат, структура й очікуваний результат запитів, що дозволяло уникнути конфліктів при злитті функціоналу.

Не менш важливою практикою стало документування процесів. Усі ключові рішення щодо архітектури, структури даних, інтерфейсів API, приклади запитів і відповіді були зафіксовані в спільній документації, яка

регулярно оновлювалася. Це забезпечило доступність знань усім членам команди й суттєво полегшило адаптацію нових розробників до проєкту.

4.3 Результати застосування адаптованої моделі життєвого циклу у плануванні ІТ проєкту

Впровадження адаптованої моделі життєвого циклу на основі методології Crystal Clear дало змогу покращити організацію роботи над проєктом вебзастосунку для прогнозу погоди. Завдяки переходу до структурованішої та більш гнучкої системи планування вдалося подолати труднощі, пов’язані з неефективною взаємодією команд, непрозорим розподілом задач і затримками, спричиненими технічними боргами.

На графіку динаміки виконання задач відображено зміну кількості завершених задач протягом усього періоду розробки. Продуктивність на етапах Т1-Т3 була відносно високою, проте характеризувалася хаотичністю та дублюванням роботи. Учасники команди не завжди узгоджували дії між собою, що призводило до паралельного виконання однакових або взаємозалежних завдань без відповідної комунікації (рисунок 4.1).

Падіння продуктивності на етапі Т4 було зумовлене накопиченням технічного боргу: невиправлені баги, відсутність тестування, погана документація та неврегульовані залежності між модулями. Це спричинило зниження темпу розробки і потребу у внутрішній реорганізації.

З Т5 команда повністю перейшла на практики методології Crystal Clear, що включали щоденні синхронізації, системне рев’ю коду, узгодження API і чіткі чеклисти якості. Це дозволило стабілізувати процес та забезпечити поступове зростання продуктивності, про що свідчить підвищення кількості завершених задач у подальших спринтах.

Рисунок 4.1 - Графік динаміки виконання задач

Також, згідно з діаграмою Ганта, ключові модулі MVP були реалізовані в межах запланованих термінів (рисунок 4.2). Починаючи з 5-го тижня, завдання стали синхронізованими між фронтендом і бекендом. Це стало можливим завдяки чіткому плануванню і щотижневим демонстраціям результатів. Наявність регулярного зворотного зв’язку від ментора дозволяла вчасно виявляти проблеми та адаптувати завдання до змін у вимогах або технічних обмеженнях.

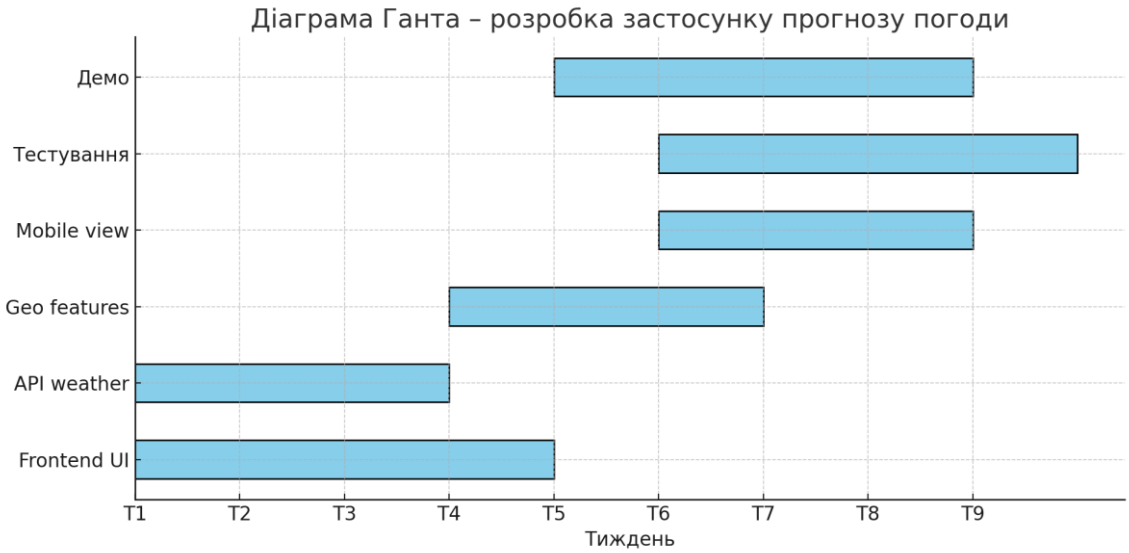

Рисунок 4.2 - Діаграмма Ганта до проєкту

Усі критично важливі компоненти застосунку - геолокація, інтеграція з OpenWeatherMap API, локалізація інтерфейсу, кросбраузерна адаптація - були завершені вчасно. Також вдалось забезпечити регулярне тестування функціоналу перед демо, що значно покращило якість проміжних релізів.

Для наочності змін, що відбулись після переходу до адаптованої моделі Crystal Clear, доцільно розглянути приклади конкретних проблем і рішень у вигляді табличного аналізу (таблиця 4.2).

Таблиця 4.2 - Приклад адаптації практик Crystal для вирішення проєктних проблем

Проблема (до, Kanban)	Рішення (Crystal)	Результат
1	2	3
Макет мобільного інтерфейсу не оновлювався 5 днів	Введено статус "Очікує залежність", дизайнер залучений до щоденних синхронізацій	Задача завершена за 1 день

Кінець таблиці 4.2

1	2	3
Прогноз погоди відображався англійською мовою	Додано контроль локалізації на етапі рев'ю API	Інтерфейс перекладено відповідно до мови UI
Не враховувалась підтримка Safari	У чеклист завершення задач додано кросбраузерну перевірку	Виправлено верстку для Safari
Розробники реалізовували запити до API по-різному	Запроваджено єдиний формат API-документації та спільні обговорення	Скорочено час інтеграції, зменшено кількість помилок
Дизайн змінювався без попередження розробників	Встановлено правило попереднього погодження змін у UI/UX	Зменшено кількість незапланованих доробок

Ці кейси підтверджують, що застосування адаптованої моделі життєвого циклу на основі Crystal Clear сприяло не лише технічному вдосконаленню процесу, а й покращенню командної взаємодії, прозорості комунікацій та ефективності управління задачами.

4.4 Висновки до четвертого розділу

У цьому розділі було розглянуто процес впровадження та результати застосування адаптованої моделі життєвого циклу розробки програмного забезпечення, що базується на принципах методології Crystal Clear. На основі аналізу виконаних робіт, командної динаміки та результатів планування можна зробити кілька ключових висновків.

По-перше, перехід від гнучкої, але недостатньо керованої моделі Kanban до структурованої Crystal Clear дав змогу забезпечити системну організацію процесів, уникнути дублювання задач, своєчасно виявляти технічні борги та краще координувати командну роботу. Методологія дозволила встановити чіткі ролі та відповідальності, а регулярні синхронізації та перегляди коду сприяли створенню єдиного інформаційного простору для всієї команди.

По-друге, завдяки впровадженню чеклистів, автоматизованого тестування та контролю локалізації вдалося суттєво підвищити якість розробки. Було виявлено і усунено ряд проблем, які раніше залишались без уваги - наприклад, неправильна локалізація погодних даних або відсутність підтримки окремих браузерів. Регулярне рев'ю API та спільне планування фронтенд- і бекенд-розробки дозволили уникати конфліктів у взаємодії між підсистемами.

По-третє, аналіз динаміки завершення задач продемонстрував, що впровадження Crystal Clear вплинуло позитивно на продуктивність команди. Після періоду падіння активності через технічні борги, команда змогла стабілізувати темп роботи, уникнути перевантажень і досягти планових результатів за графіком. Це чітко відображено на графіку завершення задач та діаграмі Ганта, яка показує, що всі основні компоненти MVP були виконані вчасно.

Крім того, додаткові кейси «було / стало» демонструють конкретні приклади покращення робочих процесів завдяки структурованому підходу - від прискореного виконання задач до покращення узгодженості між командами дизайну, розробки та тестування.

Загалом, адаптація Crystal Clear до умов невеликої команди веброзробки виявилася успішною. Вона дозволила досягти балансу між гнучкістю та структурою, що є критично важливим для швидкої та якісної реалізації прикладного ПЗ. Результати підтверджують, що навіть у короткі терміни впровадження елементів методології може суттєво підвищити як ефективність розробки, так і якість кінцевого продукту.

ВИСНОВКИ

У магістерській кваліфікаційній роботі вирішено задачу дослідження гнучких гібридних моделей управління життєвим циклом ІТ-проєктів. У процесі розв’язання цієї задачі було досягнуто таких результатів.

Проведено систематичний огляд традиційних і гнучких моделей життєвого циклу програмних проєктів, зокрема каскадної, V-подібної, інкрементної, прототипної, спіральної, ітеративної, а також методологій Scrum, Kanban, Extreme Programming, Lean, Scrumban та Crystal. Встановлено їх основні переваги, недоліки та сфери доцільного застосування. Визначено, що жодна з класичних моделей не забезпечує достатньої адаптивності до умов високої невизначеності, що є характерним для сучасних ІТ-проєктів.

На основі аналізу сучасних тенденцій управління проєктами було сформульовано доцільність і переваги застосування гібридних моделей життєвого циклу, які поєднують чіткість традиційного планування з гнучкістю адаптивних підходів. Встановлено, що такі моделі дозволяють знижувати ризики, адаптуватися до змін вимог та підтримувати баланс між швидкістю розробки і контролем за якістю продукту.

Було проаналізовано зв’язок моделей життєвого циклу з методами управління та ключовими показниками успішності програмних проєктів. Визначено, що вибір моделі напряму впливає на строки, бюджет і якість реалізації, а також на ступінь задоволеності замовника.

Проведено порівняльний аналіз сучасних інформаційних систем і сервісів для управління ІТ-проєктами (Asana, Trello, Todoist, Jira, Teamwork), за результатами якого встановлено, що найбільш гнучким, масштабованим і функціонально розвиненим інструментом є Jira. Рекомендовано використовувати цю систему для реалізації гібридного підходу в управлінні життєвим циклом ІТ-проєктів.

На підставі проведеного дослідження сформульовано узагальнені

рекомендації щодо впровадження гнучких гібридних моделей у практику управління ІТ-проєктами різного масштабу. Визначено ключові чинники успішної інтеграції гнучких підходів, зокрема потребу в автоматизації управлінських процесів, підтримці командної взаємодії та орієнтації на поступове вдосконалення продукту.

Отримані результати можуть бути використані для підвищення рівня адаптації узагальнених гібридних моделей до індивідуальних особливостей окремих різновидів ІТ-проєктів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Методичні вказівки щодо розробки та оформлення кваліфікаційної роботи другого (магістерського) рівня вищої освіти за освітньо-науковою програмою «Управління проєктами в галузі інформаційних технологій» /Упоряд.: Петров К.Е., Левикін В.М., Чалий С.Ф., Євланов М.В., Міхнов Д.К., Міхнова А.В., Чала О.В. - Харків: ХНУРЕ, 2024. - 24 с.

2. Водоспадна модель - Вікіпедія. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/Водоспадна_модель (дата звернення: 26.04.2025).

3. V-Model - Вікіпедія. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/V-Model (дата звернення: 26.04.2025).

4. Технології проєктування програмного забезпечення. URL: http://mmsa.kpi.ua/sites/default/files/disciplines/Розробка%20і%20тестування%20програм/didkovska_m_v_testing_lecture_1.pdf (дата звернення: 26.04.2025).

5. Спіральна модель - Вікіпедія. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/Спіральна_модель (дата звернення: 27.04.2025).

6. Ітеративна та інкрементна розробка - Вікіпедія. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/Ітеративна_та_інкрементна_розробка (дата звернення: 27.04.2025).

7. Гнучка розробка програмного забезпечення - Вікіпедія. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/Гнучка_розробка_програмного_забезпечення (дата звернення: 27.04.2025).

8. Sachdeva, S. (2016). Scrum methodology. Int. J. Eng. Comput. Sci, 5(16792), 16792-16800.

9. Канбан - Вікіпедія. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/Канбан (дата звернення: 29.04.2025).

10. Екстремальне програмування - Вікіпедія. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/Екстремальне_програмування (дата звернення: 29.04.2025).

11. Lean software development - Wikipedia. *Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/Lean_software_development (date of access: 30.04.2025).

12. Scrumban - Wikipedia. *Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/Scrumban (date of access: 01.05.2025).

13. Maxym Z. Crystal Agile Framework. *Maxym Zosym*. URL: https://www.maxzosim.com/crystal-agile-framework/ (date of access: 01.05.2025).

14. Manage your team's work, projects, & tasks online • Asana. *Asana*. URL: https://asana.com/ (date of access: 02.05.2025).

15. Capture, organize, and tackle your to-dos from anywhere | Trello. *Capture, organize, and tackle your to-dos from anywhere | Trello*. URL: https://trello.com/home (date of access: 02.05.2025).

16. Todoist | A To-Do List to Organize Your Work & Life. *Todoist | A To-Do List to Organize Your Work & Life*. URL: https://www.todoist.com/ (date of access: 02.05.2025).

17. Jira | Issue & Project Tracking Software | Atlassian. *Collaboration software for software, IT and business teams*. URL: https://www.atlassian.com/software/jira (date of access: 02.05.2025).

18. Robust Project Management Software for Client Services Teams. *Project and Resource Management Software for Teams*. URL: https://www.teamwork.com (date of access: 02.05.2025).