МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

М. І. Главчев, Д. М. Главчев

УПРАВЛІННЯ ТА БІЗНЕС-АНАЛІЗ ІТ-ПРОЄКТІВ

Навчальний посібник
для студентів спеціальності
F7 «Комп'ютерна інженерія»
денної та заочної форм навчання

Харків
НТУ «ХПІ»
2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

М. І. Главчев, Д. М. Главчев

УПРАВЛІННЯ ТА БІЗНЕС-АНАЛІЗ ІТ-ПРОЄКТІВ

Навчальний посібник
для студентів спеціальності
F7 «Комп'ютерна інженерія»
денної та заочної форм навчання

Затверджено
редакційно-видавничою
радою НТУ «ХПІ»,
протокол № 1 від 19.02.2026 р.

Харків
НТУ «ХПІ»
2026

УДК 004.41:005.8
Г 52

Рецензенти:
А.А. Коваленко, проф., д-р техн. наук., завідувач кафедри електронних обчислювальних машин, Харківський національний університет радіоелектроніки;
К.А. Трубчанінова, проф., д-р техн. наук, професор кафедри транспортного зв'язку, Український державний університет залізничного транспорту

Главчев М. І.

Г 52 Управління та бізнес-аналіз ІТ-проєктів : навчальний посібник для студентів спеціальності F7 «Комп'ютерна інженерія» денної та заочної форм навчання / М. І. Главчев, Д. М. Главчев. Харків : НТУ «ХПІ», 2026. 338 с.

ISBN 978-617-05-0617-7

У навчальному посібнику розглянуто ключові теоретичні та практичні аспекти з управління та бізнес-аналізу в ІТ-проєктах. Розкрито повний життєвий цикл проєкту: від ініціації та створення техніко-економічного обґрунтування до збору вимог за допомогою User Stories, управління командою, ризиками та змінами. Наведено всі необхідні терміни та сучасні підходи. Посібник містить теоретичні та практичні матеріали для кожної теми, ілюстровані пояснення та порівняльну інформацію.

Призначено для студентів спеціальності «Комп'ютерна інженерія» та інших спеціальностей галузі «Інформаційні технології» за дисципліною «Управління та бізнес-аналіз ІТ-проєктів», а також може бути корисним при курсовому та дипломному проєктуванні і для студентів ІТ-галузі.

Іл. 134. Табл. 102. Бібліогр. 53 назви.

УДК 004.41:005.8

ISBN 978-617-05-0617-7

ЗМІСТ

ВСТУП

Курс спрямований на формування у студентів цілісного уявлення про управління ІТ-проєктами та роль бізнес-аналізу в їх успішній реалізації. Протягом навчання розглядаються як класичні підходи (Waterfall, PMBOK), так і сучасні гнучкі методології (Agile, Scrum, Kanban), а також інструменти управління ресурсами, ризиками та змінами.

Особливу увагу приділено командній динаміці, ефективній комунікації зі стейкхолдерами та практичному застосуванню бізнес-аналізу для збору та управління вимогами. На практичних заняттях студенти виконуватимуть кейси, що моделюють реальні ІТ-проєкти, використовуючи сучасні інструменти управління (Jira, Trello, Confluence, MS Project).

У результаті проходження курсу студенти отримають навички:

- ✓ планування та контролю ІТ-проєктів;
- ✓ застосування Agile та класичних методологій;
- ✓ аналізу бізнес-вимог і роботи зі змінами;
- ✓ управління командою, ресурсами та ризиками;
- ✓ використання інструментів та метрик оцінки ефективності проєктів.

Курс формує компетентності майбутніх ІТ-фахівців у сфері організації та управління проєктами, поєднуючи теоретичні основи з практичними навичками роботи над реальними завданнями.

1. ВСТУП ДО УПРАВЛІННЯ ІТ-ПРОЄКТАМИ ТА БІЗНЕС-АНАЛІЗУ

1.1. Поняття проєкту та проєктного менеджменту

ІТ-проєкт – це унікальне завдання або сукупність завдань, обмежених у часі, бюджеті та ресурсах, які спрямовані на створення нового інформаційного продукту, розробку програмного забезпечення чи надання ІТ-послуги. На відміну від рутинних операцій, які повторюються, проєкт завжди має чітко визначений початок і завершення, а його результат є унікальним.

1.1.1. Характеристики ІТ-проєкту

Ключові характеристики ІТ-проєкту:

✓ Унікальність – кожен ІТ-проєкт створює новий продукт або рішення (навіть якщо схожі проєкти вже були реалізовані, вони відрізняються вимогами, технологіями чи середовищем).

✓ Обмеженість у часі – проєкт має конкретні дедлайни та часові рамки.

✓ Обмеженість у ресурсах – бюджет, команда, інфраструктура завжди є обмеженими.

✓ Визначена мета – створення нового продукту (наприклад, мобільного застосунку, вебплатформи, ERP-системи) або послуги (наприклад, впровадження CRM, хмарна міграція).

Приклади ІТ-проєктів:

✓ Розробка мобільного застосунку для інтернет-магазину.

✓ Впровадження ERP-системи на підприємстві.

✓ Міграція даних компанії до хмарної інфраструктури.

✓ Автоматизація процесів банку за допомогою нового ПЗ.

✓ Створення чат-бота для служби підтримки клієнтів.

Відмінність від операційної діяльності (табл.1.1):

✓ Проєкт тимчасовий → має початок і кінець.

✓ Операції постійні → регулярні завдання, які забезпечують роботу системи (наприклад, щоденне обслуговування серверів).

Таблиця 1.1 – Порівняння Проєкту та Операції

Проєкт	Операція
Тимчасовий, має початок і кінець	Постійна діяльність
Створює унікальний продукт/послугу	Забезпечує підтримку поточних процесів
Має обмежені ресурси	Має виділений ресурс на регулярній основі
Приклади: створення мобільного додатку, впровадження ERP	Приклади: щоденне адміністрування серверів

Висновок. IT-проєкт – це не просто «робота над продуктом», а структурована діяльність із чітко визначеною метою, результатом і межами. Розуміння цього допомагає відрізнити проєкт від поточної діяльності й правильно застосовувати методи управління.

1.1.2. Основні поняття управління проєктами

1. PMBOK (Project Management Body of Knowledge, PMI, 2017):

Проєкт – це тимчасове підприємство, спрямоване на створення унікального продукту, послуги чи результату.

✓ «Тимчасове» означає, що проєкт має чітко визначений початок і завершення.

✓ «Унікальне» вказує, що результат відрізняється від попередніх робіт (навіть якщо проєкти подібні, їхні характеристики завжди різняться).

✓ Проєкт завершується після досягнення цілей або якщо стає зрозуміло, що вони недосяжні.

2. ISO 21500 (Guidance on Project Management, 2012):

Проєкт – це унікальний процес, який складається з координованих і контрольованих робіт, має початок і кінець, виконується для досягнення визначених цілей з урахуванням обмежень за часом, вартістю, ресурсами та якістю.

Основні відмінності поняття «проєкту» наведено у таблиці 1.2.

Таблиця 1.2 – Порівняння визначень

Джерело	Ключовий акцент	Формулювання
PMBOK	Тимчасовість і унікальність	«Тимчасове підприємство для створення унікального продукту, послуги чи результату»
ISO 21500	Координація процесів і обмеження	«Унікальний процес, що складається з координованих робіт для досягнення цілей в умовах обмежень»

Приклад застосування в ІТ:

✓ PMBOK-підхід: запуск розробки нового мобільного застосунку (тимчасова команда, унікальний продукт).

✓ ISO-підхід: упровадження ERP-системи (координовані роботи різних відділів: розробка, навчання персоналу, тестування).

1.1.3. Основні параметри проєкту: цілі, час, вартість, якість

Цілі проєкту:

✓ Формулюють очікуваний результат (створення продукту, запуск сервісу, автоматизація процесів).

✓ Цілі мають бути SMART (Specific, Measurable, Achievable, Relevant, Time-bound).

Час (Time)

✓ Усі проєкти мають дедлайни: дати початку й завершення.

✓ Затримки у строках можуть призвести до збільшення витрат або втрати конкурентних переваг.

Вартість (Cost)

✓ Включає бюджет на команду, ліцензії, обладнання, хмарні сервіси.

✓ Недооцінка витрат веде до фінансових ризиків і часто до зриву проєкту.

Якість (Quality)

✓ Показує, наскільки результат відповідає очікуванням замовника та стандартам (функціональність, надійність, продуктивність).

✓ Баланс між швидкістю, бюджетом і якістю визначає успішність проєкту.

Трикутник обмежень (Iron Triangle). Управління проєктом базується на взаємозв'язку трьох головних параметрів (рис.1.1, табл.1.3):

- ✓ Час (Time)
- ✓ Вартість (Cost)
- ✓ Якість/Обсяг робіт (Quality/Scope)

Рисунок 1.1 – Трикутник обмежень

Правило: зміна одного параметра впливає на інші два.

✓ Якщо скоротити час → або зросте вартість (потрібно більше ресурсів), або постраждає якість.

✓ Якщо зменшити бюджет → доведеться знизити якість або збільшити час.

✓ Якщо розширити обсяг → зростуть і час, і вартість.

Таблиця 1.3 – Час, вартість, якість

Змінюваний параметр	Можливий вплив на інші	Приклад в IT
Час (Time)	Або зростає вартість, або знижується якість	Скорочення строків релізу → потрібна більша команда або менше тестування
Вартість (Cost)	Або зростає час, або зменшується якість	Зниження бюджету → реліз відкладається або урізають функціонал
Якість/Обсяг (Quality/Scope)	Або зростають час і вартість	Додавання нових функцій → потребує додаткових витрат і часу

Ефективне управління проєктом полягає у пошуку балансу між трьома параметрами – часом, вартістю та якістю. Керівник проєкту має враховувати взаємозалежність обмежень і приймати рішення, які оптимізують результат для бізнесу та користувачів.

Приклад в ІТ. Розробка мобільного додатку:

✓ Якщо замовник хоче випустити додаток на 2 місяці раніше – команда має або збільшити бюджет (найняти більше розробників), або знизити якість (менше тестування).

✓ Якщо бюджет скоротили на 20 % – реліз може відкластися, або доведеться урізати функціонал.

1.1.4. Роль проєктного менеджменту в ІТ

Проєктний менеджмент (Project Management) в ІТ – це систематичне застосування знань, навичок, інструментів та методів для успішної реалізації ІТ-проєктів у межах визначених строків, бюджету та якості.

Основні функції проєктного менеджменту:

- ✓ Планування
  - ✓ Визначення цілей, обсягу робіт, строків і бюджету.
  - ✓ Формування плану спринтів (Agile) або графіку робіт (Waterfall).
- ✓ Організація
  - ✓ Формування команди (розробники, тестувальники, BA, дизайнери).
  - ✓ Розподіл ролей і відповідальності.
- ✓ Мотивація та лідерство
  - ✓ Створення позитивного середовища, яке сприяє командній роботі.
  - ✓ Залучення членів команди до прийняття рішень.
- ✓ Контроль і моніторинг
  - ✓ Відстеження прогресу, своєчасне виявлення відхилень.
  - ✓ Управління ризиками та змінами.
- ✓ Комунікація зі стейкхолдерами
  - ✓ Забезпечення прозорого обміну інформацією.
  - ✓ Баланс між очікуваннями бізнесу та технічними можливостями

команди.

Чому проєктний менеджмент важливий для ІТ:

✓ Складність ІТ-проєктів: інтеграція технологій, великий обсяг вимог, швидка зміна пріоритетів.

✓ Обмежені ресурси: треба ефективно розподіляти час і бюджет.

✓ Необхідність комунікації: залучені клієнти, розробники, тестувальники, користувачі.

✓ Високий рівень невизначеності: технології швидко змінюються, потреби користувачів еволюціонують.

Приклад з практики/ У стартапі без формального управління команда може швидко почати роботу, але без плану та чіткої координації ризик провалу дуже високий. Упровадження елементарних практик проєктного менеджменту (беклог, пріоритети, регулярні зустрічі) значно підвищує шанси на успіх.

1.2. Особливості ІТ-проєктів

До особливостей ІТ-проєктів слід відмести: вимоги, ризики та структуру.

1.2.1. Висока динамічність вимог в ІТ-проєктах

Сутність проблеми/ Вимоги до ІТ-проєктів часто змінюються вже під час їх реалізації. Це відбувається через (табл.1.4):

✓ швидкий розвиток технологій;

✓ зміну ринкових умов і конкурентів;

✓ еволюцію бізнес-моделі компанії;

✓ появу нових очікувань користувачів.

На відміну від будівництва чи виробництва, де специфікації стабільні, в ІТ-проєктах «живі вимоги» є нормою.

Вплив на управління проєктом:

✓ Затримки: постійні зміни можуть відкладати реліз.

✓ Збільшення бюджету: додаткові функції потребують більше ресурсів.

✓ Ризик невідповідності продукту: якщо зміни не керовані, продукт

може втратити цілісність.

Методи роботи з динамічними вимогами:

✓ Agile-підхід: розробка ітераціями, беклог як «живий документ».

✓ Пріоритизація (MoSCoW, WSJF): визначення, які зміни є критичними, а які можна відкласти.

✓ Прототипування і юзер-сторі: допомагають швидко перевіряти нові ідеї з користувачами.

✓ Change management: формалізація процесу внесення змін (заявка → аналіз → узгодження → реалізація).

Таблиця 1.4 – Причини змін у вимогах

Причина змін	Можливі наслідки	Інструменти реагування
Швидкий розвиток технологій	Потреба інтегрувати нові рішення, переписувати код	Agile-ітерації, прототипування
Зміна ринкових умов	Зсув пріоритетів, зміна бізнес-моделі	Пріоритизація, беклог
Нові очікування користувачів	Додаткові функції, затримка релізу	Юзер-сторі, тестування з користувачами
Форс-мажор (законодавчі зміни)	Необхідність термінових змін у продукті	Change management

Приклад з практики. У стартапі команда планувала створити застосунок лише для бронювання квитків. Після перших тестів користувачі висловили потребу в онлайн-оплаті та push-сповіщеннях. Це призвело до зміни пріоритетів і збільшення бюджету, але продукт вийшов більш конкурентоспроможним.

1.2.2. Технологічні ризики та залежність від інновацій

Сутність технологічних ризиків. ІТ-проєкти реалізуються в умовах швидкої зміни технологій. Використання нових інструментів чи фреймворків завжди пов’язане з ризиками (табл.1.5):

✓ нестабільність технології (бета-версії, відсутність підтримки);

✓ залежність від вендора чи стороннього API;

✓ проблеми сумісності з іншими системами;

✓ відсутність фахівців із потрібними компетенціями.

Залежність від інновацій:

✓ Постійна потреба інтегрувати нові технології (наприклад, AI, блокчейн, хмарні рішення).

✓ Компанії, які не впроваджують інновацій, ризикують втратити конкурентоспроможність.

✓ Водночас раннє впровадження «сирих» технологій може призвести до збоїв і фінансових втрат.

Методи зниження технологічних ризиків:

✓ Proof of Concept (PoC): перевірка життєздатності технології на невеликому прототипі.

✓ Пілотні впровадження: тестування рішення в обмеженому масштабі перед розгортанням.

✓ Аналіз зрілості технології (Technology Readiness Level, TRL).

✓ Диверсифікація: уникати повної залежності від одного постачальника чи інструменту.

✓ Навчання та розвиток персоналу: інвестування в підвищення кваліфікації команди.

Таблиця 1.5 – Технологічні ризики та способи мінімізації

Тип ризику	Прояв у проєкті	Метод мінімізації
Нестабільність технології	Бета-версії, відсутність підтримки	PoC, пілотні проєкти
Залежність від вендора	Збої через сторонній API	Диверсифікація, план B
Сумісність систем	Конфлікти з існуючими рішеннями	Тестування інтеграцій
Брак компетенцій	Немає спеціалістів для нової технології	Навчання, найм експертів

Приклад з практики. Фінтех-компанія інтегрувала блокчейн для обробки транзакцій. На етапі PoC технологія виглядала перспективно, але при масштабуванні виявилися проблеми з продуктивністю. Компанія застосувала гібридний підхід: критичні операції залишилися у традиційній базі даних, а блокчейн використовувався лише для верифікації транзакцій.

1.2.3. Командна структура в ІТ-проєктах

Загальна характеристика. ІТ-проєкти реалізуються крос-функціональними командами, де кожен учасник відповідає за свою частину робіт. Важливо, щоб усі ролі взаємодіяли ефективно, адже від цього залежить швидкість, якість і кінцевий результат проєкту (табл.1.6).

Основні ролі:

- ✓ Розробники (Developers)
  - ✓ Front-end – відповідають за інтерфейс користувача, зручність і дизайн у браузері або мобільному застосунку.
  - ✓ Back-end – працюють із бізнес-логікою, базами даних, API та безпекою.
  - ✓ Full-stack – універсальні спеціалісти, що поєднують front-end і back-end навички.
- ✓ Тестувальники (QA Engineers)
  - ✓ Забезпечують якість продукту.
  - ✓ Виконують ручне та автоматизоване тестування.
  - ✓ Виявляють помилки, складають звіти й перевіряють виправлення.
- ✓ DevOps-інженери
  - ✓ Автоматизують процеси розгортання, інтеграції та доставки продукту (CI/CD).
  - ✓ Підтримують інфраструктуру (сервери, хмарні сервіси).
  - ✓ Відповідають за стабільність і масштабованість системи.
- ✓ Бізнес-аналітики (BA)
  - ✓ Збирають вимоги від замовника.
  - ✓ Перекладають бізнес-цілі у технічні завдання для команди.
  - ✓ Контролюють відповідність продукту очікуванням користувачів.

- ✓ Проєктний менеджер (PM)
  - ✓ Планує та координує роботу команди.
  - ✓ Відповідає за строки, бюджет і комунікацію зі стейкхолдерами.
  - ✓ Розв’язує конфлікти й знімає блокери.

Таблиця 1.6 – Командна структура: ролі та їх значення

Роль	Основні завдання	Значення для проєкту
Front-end розробник	Розробка інтерфейсу користувача, робота з HTML, CSS, JavaScript, React, Angular.	Забезпечує зручність і доступність продукту для користувачів.
Back-end розробник	Робота з бізнес-логікою, базами даних, API, безпекою.	Гарантує функціональність та стабільність роботи системи.
Full-stack розробник	Поєднання завдань front-end і back-end.	Універсальність, можливість закривати прогалини в команді.
QA інженер	Тестування продукту (ручне й автоматизоване), звіти про помилки.	Забезпечує якість і надійність ПЗ.
DevOps інженер	Автоматизація CI/CD, підтримка інфраструктури, масштабованість.	Гарантує стабільність системи та швидке оновлення продукту.
Бізнес-аналітик	Збір і формалізація вимог, трансформація бізнес-цілей у технічні завдання.	Забезпечує відповідність продукту потребам клієнта.
Проєктний менеджер	Планування, координація команди, контроль бюджету й строків.	Організовує роботу та підтримує баланс між стейкхолдерами й командою.

Приклад взаємодії ролей. У проєкті з розробки мобільного банківського застосунку:

- ✓ BA збирає вимоги від банку.
- ✓ Front-end створює інтерфейс.
- ✓ Back-end розробляє API для транзакцій.
- ✓ QA перевіряє коректність платежів.
- ✓ DevOps автоматизує деплой у хмару.
- ✓ PM стежить за графіком і бюджетом.

1.3. Міжнародний характер ІТ-проєктів

Сутність явища. Багато сучасних ІТ-проєктів реалізуються у глобальному масштабі:

- ✓ замовник у США, команда розробки в Україні, тестування в Індії, клієнти в Європі;
- ✓ використання міжнародних платформ, сервісів і стандартів.

Це створює як переваги, так і виклики для управління.

Переваги міжнародності:

- ✓ Доступ до талантів: можна залучити найкращих спеціалістів з усього світу.
- ✓ Цілодобова розробка: завдяки різниці в часових зонах команда може працювати майже безперервно.
- ✓ Різноманітність ідей: мультикультурні команди приносять різні підходи й рішення.

Виклики міжнародних ІТ-проєктів:

- ✓ Часові зони: складність планування зустрічей.
- ✓ Мовні бар'єри: непорозуміння при формулюванні вимог.
- ✓ Культурні відмінності: різні стилі комунікації й роботи.
- ✓ Юридичні й податкові особливості: необхідність враховувати законодавство різних країн.

Методи подолання викликів:

- ✓ Використання гнучких інструментів комунікації (Slack, Jira, Confluence, Zoom).
- ✓ Чітка документація англійською як lingua franca.
- ✓ Розподіл відповідальності за часовими зонами.
- ✓ Навчання команди міжкультурній компетентності.

Приклад з практики. Міжнародна корпорація замовила розробку CRM-системи:

- ✓ Бізнес-аналітики працювали у Лондоні,
- ✓ розробка велася у Львові,
- ✓ підтримка клієнтів – у Філіппінах.

Завдяки чітким процесам комунікації й використанню Jira проєкт був завершений у строк, попри культурні та часові відмінності.

1.4. Приклади IT-проєктів

1. Розробка вебсервісу

Опис: створення онлайн-платформи (наприклад, маркетплейсу чи системи бронювання).

Особливості:

- ✓ Велика кількість користувачів одночасно.
- ✓ Необхідність масштабованості та безпеки.
- ✓ Часті оновлення функціоналу за потребами бізнесу.
- ✓ Ризики: проблеми з продуктивністю, кіберзагрози, швидка зміна ринку.

2. Розробка мобільного додатку

Опис: створення застосунку для iOS та Android (наприклад, банкінг чи доставка їжі).

Особливості:

- ✓ Адаптація під різні платформи та пристрої.
- ✓ Високі вимоги до UX/UI.
- ✓ Необхідність регулярних оновлень у магазинах додатків.
- ✓ Ризики: низький рейтинг у App Store/Google Play, відмова користувачів через слабкий UX.

3. Впровадження корпоративної ERP-системи

Опис: інтеграція системи управління ресурсами підприємства (SAP, Oracle, Microsoft Dynamics).

Особливості:

- ✓ Складна інтеграція з існуючими бізнес-процесами.
- ✓ Необхідність навчання персоналу.

✓ Високий бюджет і довготривалий цикл реалізації.

✓ Ризики: опір змінам з боку працівників, перевищення бюджету, складність підтримки.

Таблиця 1.7 – Приклади ІТ-проєктів

Тип проєкту	Особливості	Ризики
Вебсервіс	Масштабованість, безпека, часті оновлення	Продуктивність, кіберзагрози
Мобільний додаток	UX/UI, різні платформи, регулярні оновлення	Низький рейтинг, відмова користувачів
ERP-система	Інтеграція процесів, навчання персоналу, бюджет	Опір змінам, перевищення бюджету

Висновок. Кожен тип ІТ-проєкту має власну специфіку: вебсервіси орієнтовані на масштабованість, мобільні додатки – на зручність користувача, ERP – на інтеграцію та оптимізацію бізнес-процесів.

1.5. Життєвий цикл ІТ-проєкту

Для розуміння слід розглянути фази життєвого циклу проєкту для різних моделей.

1.5.1. Моделі

1. Модель Waterfall (каскадна):

Послідовний процес (рис.1.2): аналіз → проєктування → розробка → тестування → впровадження → підтримка.

Переваги: чітка структура, добре підходить для проєктів із фіксованими вимогами.

Недоліки: слабка гнучкість, складність внесення змін.

Приклад: створення ERP-системи для держустанови.

Рисунок 1.2 – Модель Waterfall

2. V-model (верифікація і валідація):

Розширення Waterfall: кожному етапу відповідає свій етап тестування (аналіз ↔ acceptance-тести, проєктування ↔ системні тести, кодування ↔ модульні тести) (рис.1.3).

Переваги: висока увага до якості, системне тестування з самого початку.

Недоліки: ще менш гнучка модель, ніж Waterfall.

Приклад: розробка програмного забезпечення для авіації чи медицини.

Рисунок 1.3 – V-model

3. Agile-моделі:

Ітеративна й інкрементальна розробка.

Вимоги уточнюються поступово, користувачі залучаються на всіх етапах (рис.1.4).

Переваги: висока гнучкість, швидке реагування на зміни, ранній зворотний зв’язок.

Недоліки: важко прогнозувати точні строки та бюджет.

Приклад: створення мобільного застосунку для стартапу.

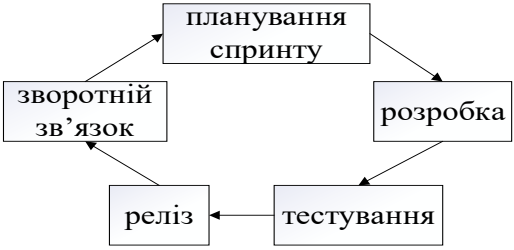

Рисунок 1.4 – Agile-моделі

Таблиця 1.8 – Порівняння моделей життєвого циклу

Модель	Переваги	Недоліки	Приклад застосування
Waterfall	Чітка структура, зрозумілі етапи, добре працює при фіксованих вимогах.	Низька гнучкість, складність внесення змін після початку реалізації.	ERP-системи для державних установ.
V-model	Сильна увага до якості, тести на кожному етапі, прозорий контроль.	Ще менш гнучка, ніж Waterfall; висока вартість змін.	Розробка ПЗ для авіації або медицини.
Agile	Висока гнучкість, адаптація до змін, залучення користувача, швидкий зворотний зв'язок.	Складно точно прогнозувати бюджет і строки.	Мобільні застосунки для стартапів.

1.5.2. Фази життєвого циклу ІТ-проєкту

1. Ініціація

- ✓ Визначення мети та обґрунтування проєкту (business case).
- ✓ Аналіз зацікавлених сторін (stakeholder analysis).
- ✓ Формування команди та призначення проєктного менеджера.
- ✓ Основний результат: Статут проєкту (Project Charter).

2. Планування

- ✓ Деталізація цілей, визначення вимог.
- ✓ Розробка плану робіт (WBS), бюджету, графіка.
- ✓ Визначення ризиків і методів їх мінімізації.
- ✓ Основний результат: Project Management Plan.

3. Виконання
- ✓ Реалізація завдань згідно з планом.
- ✓ Управління командою, комунікаціями, якістю.
- ✓ Використання інструментів (Jira, Trello, MS Project).
- ✓ Основний результат: Інкременти продукту.

4. Моніторинг і контроль
- ✓ Відстеження прогресу (KPI, метрики).
- ✓ Управління змінами та ризиками.
- ✓ Корекція планів при відхиленнях.
- ✓ Основний результат: Звіти про статус і виконання.

5. Завершення
- ✓ Передача продукту замовнику.
- ✓ Підготовка фінального звіту.
- ✓ Аналіз уроків (lessons learned).
- ✓ Основний результат: Закритий проєкт, підсумкова документація.

Таблиця 1.9 – Фази життєвого циклу

Фаза	Основні завдання	Основний результат
Ініціація	Визначення мети, аналіз стейкхолдерів, формування команди.	Статут проєкту (Project Charter)
Планування	Формування вимог, WBS, бюджет, ризики, план управління.	Project Management Plan
Виконання	Реалізація завдань, управління командою, комунікаціями та якістю.	Інкременти продукту
Моніторинг і контроль	Відстеження прогресу, управління ризиками й змінами, корекція плану.	Звіти про статус і виконання
Завершення	Передача продукту, фінальний звіт, lessons learned.	Закритий проєкт, підсумкова документація

1.5.3. Приклади застосування життєвого циклу

1. Стартап-проєкт (Agile-підхід)

✓ Ініціація: ідея мобільного додатку для доставки їжі. Засновники формують невелику команду (5-7 людей).

✓ Планування: лише високорівневий беклог із ключовими функціями (замовлення, оплата, геолокація). Деталізація вимог відбувається поступово.

✓ Виконання: короткі спринти по 2 тижні, швидкі прототипи, постійний контакт із тестовими користувачами.

✓ Моніторинг і контроль: щоденні стендапи, ретроспективи, швидка зміна пріоритетів у беклозі.

✓ Завершення: MVP випущено через 3 місяці, далі – нові спринти з додаванням функцій.

Особливість: висока гнучкість, орієнтація на швидкий вихід на ринок, ризик невизначеності.

2. Корпоративний проєкт (Waterfall/V-model)

✓ Ініціація: банк вирішує впровадити нову ERP-систему для управління фінансами. Формується велика проєктна команда (50+ осіб).

✓ Планування: складається детальний план на 1,5 роки, бюджет у мільйони доларів. Усі вимоги документуються у специфікації.

✓ Виконання: етапи йдуть послідовно: аналіз → проєктування → розробка → тестування.

✓ Моніторинг і контроль: регулярні звіти керівництву, формальні контрольні точки.

✓ Завершення: після інтеграції та тестування система впроваджується одноразово для всіх відділів.

Особливість: високий рівень формалізації, великі бюджети, низька гнучкість у зміні вимог.

Таблиця 1.10 – Порівняння прикладів життєвого циклу

Критерій	Стартап (Agile)	Корпоративний проєкт (Waterfall/V-model)
Тривалість фаз	Короткі, ітеративні	Довгі, послідовні
Вимоги	Динамічні, уточнюються	Фіксовані на початку

	поступово	
Команда	Мала, крос-функціональна	Велика, спеціалізована
Моніторинг	Щоденні зустрічі, ретроспективи	Регулярні звіти, контрольні точки
Ризики	Висока невизначеність	Перевищення бюджету, опір змінам

1.5.4. Порівняння гнучких і традиційних підходів у життєвому циклі

У таблиці 1.11 наведена порівняння гнучких і традиційних підходів у життєвому циклі проєкту.

Таблиця 1.11 – Порівняння підходів у життєвому циклі проєкту

Фаза	Традиційний підхід (Waterfall, V-model)	Гнучкий підхід (Agile, Scrum, Kanban)
Ініціація	Докладне ТЗ формується на старті, затверджується контрактом.	Формулюється загальне бачення та беклог із високорівневими вимогами.
Планування	Детальний план на весь проєкт (WBS, бюджети, графіки).	Планування ітерацій (спринтів), постійне уточнення беклогу.
Виконання	Послідовне виконання фаз; продукт формується ближче до кінця.	Ітеративна розробка; працюючі інкременти вже з перших спринтів.
Моніторинг і контроль	Формальні контрольні точки, звіти, відповідність плану.	Щоденні стендапи, ретроспективи, швидке реагування на зміни.
Завершення	Продукт передається наприкінці життєвого циклу.	Кожна ітерація дає робочий продукт; завершення може бути гнучким.

Висновки:

✓ Традиційні підходи більше підходять для стабільних середовищ (державні та критичні проєкти).

✓ Гнучкі підходи – для динамічних ринків (стартапи, продуктові компанії).

✓ Вибір залежить від природи проєкту, і в сучасній практиці часто використовують гібридні моделі.

1.6. Бізнес-аналіз у контексті ІТ-проєктів

Business Analysis Body of Knowledge (BABOK Guide) – міжнародний стандарт, розроблений International Institute of Business Analysis (IIBA).У ньому бізнес-аналіз визначається як: «Практика розуміння потреб бізнесу та визначення рішень, які приносять цінність зацікавленим сторонам» (рис.1.5)

1.6.1. Бізнес-аналіз як складова управління ІТ-проєктами

Ключові аспекти визначення:

✓ Фокус на потребах бізнесу – аналітик виявляє, що саме потрібно організації або клієнту, а не лише формулює технічні вимоги.

✓ Орієнтація на цінність – рішення має приносити вимірювану користь (збільшення прибутку, оптимізація витрат, покращення сервісу).

✓ Робота із зацікавленими сторонами – аналітик діє як посередник між бізнесом та технічними фахівцями.

✓ Не обмежується ІТ – бізнес-аналіз може охоплювати зміни у процесах, організаційній структурі, політиках компанії.

Чому це важливо в ІТ-проєктах?

✓ ІТ-рішення завжди є засобом для досягнення бізнес-цілей.

✓ Без правильного бізнес-аналізу команда ризикує створити продукт, який технічно якісний, але не вирішує ключових проблем користувачів.

✓ Наприклад, розробка CRM без урахування потреб відділу продажів призведе до того, що система не буде використовуватися.

Рисунок 1.5 – Бізнес-аналіз у ІТ-проєктів

1.6.2. Роль бізнес-аналітика як посередника

Посередницька функція:

✓ Бізнес-аналітик (BA) виступає зв'язуючою ланкою між замовником (бізнесом, стейкхолдерами) та командою розробки.

✓ Він «перекладає» мову бізнесу (цілі, потреби, цінність) на мову технічних вимог (юзер-сторі, діаграми, специфікації).

✓ Забезпечує двосторонню комунікацію:

✓ бізнес отримує прозорість у тому, що саме і як створюється,

✓ команда має чітке уявлення, чому ці функції важливі.

Типові завдання BA як посередника:

✓ Збирає потреби від замовника та кінцевих користувачів.

✓ Формалізує їх у вигляді бізнес-вимог і технічних специфікацій.

✓ Допомагає замовнику зрозуміти обмеження ресурсів, часу і технологій.

✓ Пояснює команді бізнес-цінність функцій, щоб уникнути формального «програмування без контексту».

✓ Узгоджує зміни в проєкті, щоб вони не суперечили загальній стратегії.

Приклад:

✓ Замовник: «Мені потрібен зручний застосунок для замовлення таксі».

✓ BA: формалізує вимогу у вигляді функцій – «пошук авто за геолокацією», «оплата карткою», «push-сповіщення».

✓ Команда: отримує чіткі задачі у Jira й розуміє, що їхня робота напряму впливає на зручність користувача та прибуток бізнесу.

Візуалізація ролі BA наведена на рис.1.6.

Рисунок 1.6 – Роль BA

1.6.3. Інструменти бізнес-аналізу в ІТ-проєктах

Опис інструментів. Нижче наведені основні інструменти (табл.1.12), які бізнес-аналітики застосовують у практиці ІТ-проєктів:

Таблиця 1.12 – Інструменти бізнес-аналізу

Інструмент	Призначення	Приклад
User Stories	Формулювання вимог від імені користувача	«Як клієнт, я хочу…»
Use Case Diagram	Візуалізація взаємодії користувачів із системою	«Користувач → Переглянути статистику»
Прототипи	Швидка візуалізація інтерфейсу та сценаріїв	Прототип мобільного застосунку у Figma
Backlog	Список завдань і вимог для управління розробкою	Елементи в Jira: «Фільтрація товарів», «Оплата»

1. User Stories (користувацькі історії)

✓ Формат:

Як [роль], я хочу [дія], щоб [цінність].

✓ Приклад: «Як користувач мобільного банкінгу, я хочу отримувати push-сповіщення про транзакції, щоб швидко контролювати рух коштів».

✓ Використання: допомагають формувати вимоги у зрозумілому бізнесу форматі та створюють основу для тест-кейсів.

2. Use Case Diagram (діаграма варіантів використання, UML)

✓ Показує, як користувачі (актор) взаємодіють із системою.

✓ Дає цілісне уявлення про функціональність системи.

✓ Приклад: актори «Адміністратор» та «Користувач» і варіанти – «Створити акаунт», «Редагувати профіль», «Переглянути статистику».

3. Прототипи

✓ Візуальні ескізи або інтерактивні макети інтерфейсу.

✓ Дають можливість замовнику і команді швидко узгодити вигляд і логіку продукту ще до початку розробки.

✓ Інструменти: Figma, Balsamiq, Axure.

✓ Приклад: прототип екрану мобільного застосунку для замовлення їжі з кнопками «Меню», «Кошик», «Оплатити».

4. Backlog (беклог продукту/проєкту)

✓ Список усіх завдань і вимог до системи, який постійно оновлюється.

✓ Містить як функціональні, так і нефункціональні вимоги.

✓ Використовується у Scrum/Kanban для планування спринтів і відстеження прогресу.

✓ Приклад: у Jira беклог може містити історії «Фільтрація товарів», «Додавання в кошик», «Авторизація через Google».

1.6.4. Значення бізнес-аналізу в IT-проєктах

Бізнес-аналіз, як інструмент зниження ризиків:

✓ Ризик невідповідності продукту очікуванням клієнта → BA формалізує вимоги, перевіряє їх із замовником, уточнює пріоритети.

✓ Ризик перевищення бюджету і строків → BA допомагає визначити критично важливі функції (MVP), а менш важливі відкласти.

✓ Ризик технічної несумісності → BA разом із архітекторами узгоджує бізнес-вимоги з технічними можливостями.

Внесок у забезпечення успіху:

✓ Ясність цілей → BA забезпечує, щоб команда мала спільне бачення кінцевого результату.

✓ Пріоритизація → завдяки бізнес-аналізу команда працює над тим, що приносить найбільшу цінність.

✓ Комунікація → BA зменшує кількість непорозумінь між бізнесом і технічними фахівцями.

✓ Якість продукту → чіткі вимоги дозволяють побудувати тест-кейси,

що перевіряють критично важливі сценарії.

Приклади з практики:

✓ Стартап: завдяки аналізу користувацьких історій команда сфокусувалася на ключових функціях додатку для доставки їжі (замовлення і оплата), замість перевантаження другорядними. Це дозволило випустити продукт вчасно.

✓ Корпорація: під час впровадження ERP BA допоміг уникнути провалу, бо вчасно виявив конфлікт у вимогах від фінансового й логістичного відділів.

Висновок. Бізнес-аналіз – це страхувальна сітка в управлінні ІТ-проєктами: він не лише документує вимоги, а й допомагає передбачити та мінімізувати ризики, забезпечуючи тим самим успішність проєкту.

1.7. Загальні висновки

1.7.1. Ключовий підсумок

1. Що таке ІТ-проєкт – його унікальність, обмеження часу, ресурсів і сфери.
2. Основні поняття управління проєктами – визначення за PMBOK та ISO 21500, «трикутник обмежень».
3. Особливості ІТ-проєктів – динамічність вимог, технологічні ризики, командна структура.
4. Життєвий цикл ІТ-проєкту – моделі (Waterfall, V-model, Agile), фази (ініціація → планування → виконання → моніторинг → завершення).
5. Бізнес-аналіз у проєктах – визначення за BABOK, роль BA як посередника, інструменти (User Stories, Use Case Diagram, прототипи, backlog).
6. Значення бізнес-аналізу – зменшення ризиків, забезпечення цінності та успіху проєкту.

1.7.2. Питання для самоперевірки

1. Чим ІТ-проєкт відрізняється від операційної діяльності?
2. Що таке «трикутник обмежень» і які його складові?

3. Чому IT-проєкти мають високу динамічність вимог?
4. Які основні фази життєвого циклу проєкту і чим вони відрізняються?
5. У чому різниця між Waterfall та Agile на рівні життєвого циклу?
6. Як BABOK визначає бізнес-аналіз?
7. Чому бізнес-аналітика називають «посередником»?
8. Які інструменти бізнес-аналізу ви знаєте і які їхні приклади?
9. Як бізнес-аналіз допомагає зменшити ризики перевищення бюджету й строків?
10. Який приклад із практики показує, що BA може вплинути на успіх проєкту?

1.8. Контрольні запитання

1. Що таке IT-проєкт? Назвіть його ключові характеристики та наведіть приклади.
2. У чому полягає відмінність між проєктом та операційною діяльністю? Поясніть на прикладі IT-галузі.
3. Як PMBOK визначає поняття «проєкт»? Що означає «тимчасовість» і «унікальність» у цьому контексті?
4. Яке визначення проєкту пропонує стандарт ISO 21500 і чим воно відрізняється від визначення PMBOK?
5. Що таке «трикутник обмежень» (Iron Triangle) у проєктному менеджменті? Опишіть взаємозв'язок між його параметрами.
6. Як зміна одного з параметрів (час, вартість, якість) впливає на інші? Наведіть конкретний приклад із розробки ПЗ.
7. Що таке SMART-цілі проєкту? Розшифруйте абревіатуру та поясніть кожен критерій.
8. Які моделі життєвого циклу IT-проєкту ви знаєте? Коротко охарактеризуйте кожну.
9. Охарактеризуйте основні фази життєвого циклу IT-проєкту та їх зміст.
10. У чому полягає роль бізнес-аналізу в контексті управління IT-проєктами?

11. Які технологічні ризики характерні для ІТ-проєктів і чому вони зумовлені залежністю від інновацій?

12. Яка командна структура типова для ІТ-проєктів? Охарактеризуйте основні ролі.

13. Чому ІТ-проєкти мають міжнародний характер і які особливості це породжує для управління?

14. Як бізнес-аналітик виступає посередником між бізнесом і технічною командою? Наведіть приклад.

15. Порівняйте гнучкі та традиційні підходи у контексті життєвого циклу ІТ-проєкту: переваги та недоліки.

2. КЛАСИЧНІ ПІДХОДИ ДО УПРАВЛІННЯ ІТ-ПРОЄКТАМИ

У цій лекції розглянемо класичні підходи до управління ІТ-проєктами. Вони виникли ще у 60-70-х роках XX століття і до сьогодні залишаються актуальними. Основна ідея традиційних методів – це передбачуваність, формалізація та контроль.

Незважаючи на домінування Agile у багатьох стартапах і комерційних продуктах, Waterfall та PMBOK широко застосовуються у сферах, де важливі точність, стабільність та контрактні зобов'язання (державні проєкти, банківська сфера, оборонні системи).

2.1. Вступ

Традиційні методи управління проєктами з'явилися значно раніше, ніж сучасні гнучкі підходи. Вони формувалися в другій половині XX століття в умовах, коли бізнес і державні структури прагнули до максимальної формалізації, стандартизації та передбачуваності результатів. Тоді помилка могла коштувати не лише фінансових втрат, але й людських життів – особливо у військових, авіаційних чи космічних програмах.

2.1.1. Актуальність класичних методів в управлінні проєктами

Класичні методи управління (зокрема, Waterfall та PMBOK) забезпечували високу керованість і контрольованість усіх процесів, дозволяли фіксувати обсяг робіт ще на початку та контролювати кожен наступний етап. Завдяки цьому такі методи широко застосовувались і залишаються актуальними там, де важливі:

- ✓ суворе дотримання термінів і бюджетів;
- ✓ відповідність нормативним і юридичним вимогам;
- ✓ контроль якості та безпеки;
- ✓ стабільність вимог упродовж усього життєвого циклу проєкту.

Сучасна практика показує, що попри значну популярність Agile, класичні методи не лише зберегли свою роль, але й у багатьох випадках залишаються безальтернативними.

2.1.2. Чому, незважаючи на розвиток Agile, Waterfall і PMBOK досі залишаються затребуваними

На перший погляд може здатися, що розвиток Agile повністю витіснив традиційні методології. Адже Agile дозволяє швидко адаптуватися до змін, скорочує час виходу продукту на ринок і зменшує ризик невідповідності кінцевого результату очікуванням замовника.

Проте реальність значно складніша. У багатьох сферах проєкти мають такі особливості, що роблять використання Agile малоефективним або навіть небезпечним:

✓ Жорсткі нормативні обмеження. У сфері охорони здоров'я, фінансів чи оборони кожна зміна продукту вимагає сертифікації й тривалої перевірки. Часті зміни, характерні для Agile, у таких умовах неможливі.

✓ Великий масштаб проєкту. Гігантські проєкти з бюджетом у сотні мільйонів доларів потребують чіткого плану і прогнозованості. Невизначеність Agile у цьому випадку може призвести до катастрофічних ризиків.

✓ Фокус на документації. У проектах, що реалізуються десятками підрядників і сотнями людей, саме документація стає ключем до координації роботи. Agile ж більше орієнтований на комунікацію в команді.

✓ Довгострокові державні програми. Багато урядових чи військових проєктів плануються на десятки років наперед. Тут необхідна не швидкість, а стабільність і контроль.

Таким чином, Waterfall і PMBOK залишаються затребуваними тому, що вони відповідають потребам масштабних, критично важливих і регламентованих проєктів, де швидкі зміни – не перевага, а ризик.

2.1.3. Приклади галузей, де переважають традиційні підходи

1. Банківські системи.

Банківський сектор вимагає максимальної надійності й відповідності міжнародним стандартам (наприклад, PCI DSS). Будь-які зміни у фінансовому ПЗ потребують тривалого погодження і тестування, тому Agile тут застосовується обмежено. Більшість основних банківських платформ розроблялися саме за Waterfall або на основі PMBOK.

2. Державні проєкти.

Державні тендери майже завжди передбачають фіксований бюджет, чіткі терміни та детальну документацію. Наприклад, системи електронного врядування, податкові або митні системи здебільшого створюються за каскадними моделями, оскільки кожна зміна вимог може потребувати нового узгодження із замовником.

3. Військові розробки.

У військових і космічних проєктах пріоритетом є безпека, перевірюваність і стандартизація. Програми NASA чи системи ППО проектуються й тестуються за суворими правилами, які неможливо змінювати «на льоту». Тут будь-який «гнучкий» підхід є небезпечним.

У таблиці 2.1 наведено порівняння сфер застосування.

Таблиця 2.1 – Сфери застосування методів

Сфера	Класичні методи (Waterfall і PMBOK)	Agile
Банківські системи	так	Обмежено
Державні проєкти	так	Майже не застосовується
Військові розробки	так	Непридатне
Комерційні продукти	Частково	так
Стартапи	Рідко	так

Таким чином, класичні підходи не є архаїчними – вони залишаються незамінними в умовах, де гнучкість і швидкість є менш важливими, ніж стабільність, безпека та прогнозованість.

2.2. Модель Waterfall

Waterfall – класична каскадна модель управління проєктами.

2.2.1. Історія виникнення (Б. Боем, 1970-ті роки)

Каскадна модель розробки програмного забезпечення, відома як Waterfall, виникла на початку 1970-х років. Хоча елементи послідовного підходу використовувалися й раніше, саме в цей період модель отримала

свою назву та чітке формальне описання.

Перші згадки про каскадну модель можна знайти у працях доктора Вінстона Ройса (Winston W. Royce), американського інженера-програміста, який у 1970 році опублікував статтю «Managing the Development of Large Software Systems». У цій статті він запропонував ілюстративну схему, яка нагадувала водоспад: кожна стадія спускалася вниз до наступної, як вода, що стікає каскадами.

Цікаво, що сам Ройс не рекомендував використовувати Waterfall у «чистому вигляді». У своїй статті він зазначав, що жорстке послідовне проходження етапів є занадто ризикованим для складних програмних проєктів. Він пропонував додавати проміжні перевірки й зворотні зв'язки між етапами. Проте саме спрощене розуміння його схеми було активно підхоплене практиками, і модель отримала назву «водоспадна» (Waterfall).

У 1970-1980-ті роки цей підхід став домінуючим у світі розробки програмного забезпечення. Причин було декілька:

✓ Схожість із класичними інженерними процесами. У будівництві чи машинобудуванні вже давно використовувалася поетапна організація роботи: спочатку проектування, потім будівництво, потім перевірка. ІТ-фахівці просто адаптували ці підходи.

✓ Вимоги замовників. Великі державні та військові організації (зокрема Міністерство оборони США та NASA) потребували максимальної передбачуваності й контролю.

✓ Недостатність досвіду в управлінні програмними проєктами. У ті роки розробка ПЗ була новою галуззю, і методології лише формувалися. Чітка каскадна модель виглядала як єдиний зрозумілий спосіб організувати роботу.

Особливу роль у поширенні Waterfall відіграв американський науковець Баррі Боем (Barry Boehm). У 1980-ті роки він активно займався вдосконаленням моделей життєвого циклу ПЗ та опублікував роботу «A Spiral Model of Software Development and Enhancement» (1986), де розкритикував жорсткий Waterfall і запропонував більш гнучку спіральну модель. Проте саме завдяки його працям Waterfall остаточно закріпився як базовий приклад «класичного» підходу.

Отже, Waterfall – це метод, що сформувався як логічне продовження інженерних традицій у 1970-ті роки. Він став першою по-справжньому формалізованою методологією у сфері програмування і на десятиліття визначив розвиток проектного менеджменту в IT (рис.2.2).

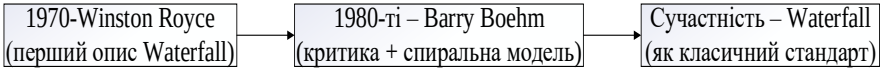

Рисунок 2.2 – Еволюція каскадної моделі

2.2.2. Основні фази каскадної моделі

2.2.2.1. Аналіз вимог

Фаза аналізу вимог є фундаментом каскадної моделі. Вона визначає, що саме потрібно створити і яким має бути кінцевий продукт. У цій фазі формуються ключові артефакти, які впливають на весь подальший життєвий цикл проєкту.

Мета етапу:

✓ Виявити всі потреби замовника й кінцевих користувачів.

✓ Зафіксувати вимоги у формі, зрозумілій як технічним спеціалістам, так і бізнес-замовникам (рис.2.3).

✓ Узгодити межі проєкту: що входить і що не входить до його складу.

Процес аналізу вимог включає (рис.2.3):

1. Збір вимог

✓ Інтерв'ю зі стейкхолдерами.

✓ Анкетування та опитування кінцевих користувачів.

✓ Аналіз бізнес-процесів.

✓ Вивчення аналогів та конкурентів.

2. Документування вимог. Результати оформлюються у вигляді:

✓ SRS (Software Requirements Specification) – специфікація вимог до програмного забезпечення (табл.2.2).

✓ Діаграм (UML, BPMN), які показують взаємозв'язки між процесами та системою.

✓ Переліку нефункціональних вимог: продуктивність, безпека,

надійність.

3. Аналіз і узгодження
- ✓ Перевірка на суперечності.
- ✓ Оцінка повноти й узгодженість вимог.
- ✓ Затвердження документів із замовником.

4. Вихідні артефакти етапу
- ✓ SRS (основний документ).
- ✓ Протоколи зустрічей і погоджень.
- ✓ Попередні діаграми системи.

Рисунок 2.3 – Схема процесу аналізу вимог

Таблиця 2.2 – Приклад структури SRS

Розділ	Зміст
Вступ	Мета, область застосування
Загальний опис	Опис системи та користувача
Функціональні вимоги	Список функцій, сценарій використання
Нефункціональні вимоги	Продуктивність, безпека, сумустність
Додатки	Діаграми, глосарій

Приклад. Уявімо, що банк замовляє систему інтернет-банкінгу. На етапі аналізу вимог фіксується:

✓ функціональні вимоги: авторизація користувачів, перегляд балансу, переказ коштів;

✓ нефункціональні: час відгуку не більше 2 секунд, підтримка навантаження до 50 тис. користувачів одночасно, відповідність стандарту PCI DSS.

Без чіткого аналізу вимог подальший розвиток такого проєкту є неможливим.

Проблеми етапу аналізу вимог

✓ Замовники часто не до кінця розуміють, чого саме хочуть.

✓ Вимоги суперечливі або нечіткі.

✓ Небезпека «замороження» – поки триває збір вимог, технології та ринок можуть змінитися.

2.2.2.2. Проєктування системи

Значення етапу проєктування.

Етап проєктування – це міст між вимогами та реалізацією. Якщо під час аналізу ми відповідаємо на запитання «що потрібно створити?», то на етапі проєктування визначається «як саме це реалізувати?».

Саме тут формуються архітектурні рішення, що забезпечують подальшу розробку (рис.2.4). У традиційній каскадній моделі цей етап вважається одним із ключових, адже правильність і повнота проєктної документації безпосередньо впливають на успішність усієї системи.

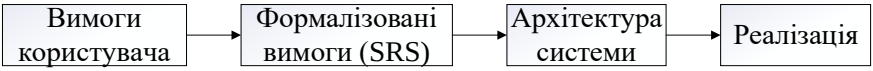

Рисунок 2.4 – Схема «Шлях від вимог до архітектури»

Основні завдання проєктування:

1. Вибір архітектури системи. Визначається, чи буде система централізованою чи розподіленою, які компоненти матиме та як вони взаємодіятимуть.

2. Проєктування бази даних. Створюються логічні та фізичні моделі даних, ER-діаграми, визначаються ключі та зв’язки між сутностями.

3. Вибір технологій та інструментів. На цьому етапі приймається рішення щодо мов програмування, фреймворків, платформ і серверів.

4. Розробка технічної документації. Створюються діаграми (UML, DFD), специфікації інтерфейсів, описи алгоритмів.

Артефакти етапу проєктування

✓ Технічне завдання (ТЗ) або деталізований проєктний документ.

✓ Архітектурні діаграми. Наприклад:

✓ діаграма компонентів;

✓ ER-діаграма бази даних;

- ✓ діаграма класів (UML).
- ✓ Специфікації модулів – опис логіки, інтерфейсів, API.

Приклад. Якщо на попередньому етапі для банківської системи визначено, що потрібні модулі «Авторизація», «Платежі», «Звіти», то під час проєктування:

- ✓ розробляється архітектура клієнт-сервер;
- ✓ проектується база даних користувачів, рахунків і транзакцій;
- ✓ визначається, що модуль «Платежі» взаємодіятиме з процесинговим центром через API (рис.2.5).

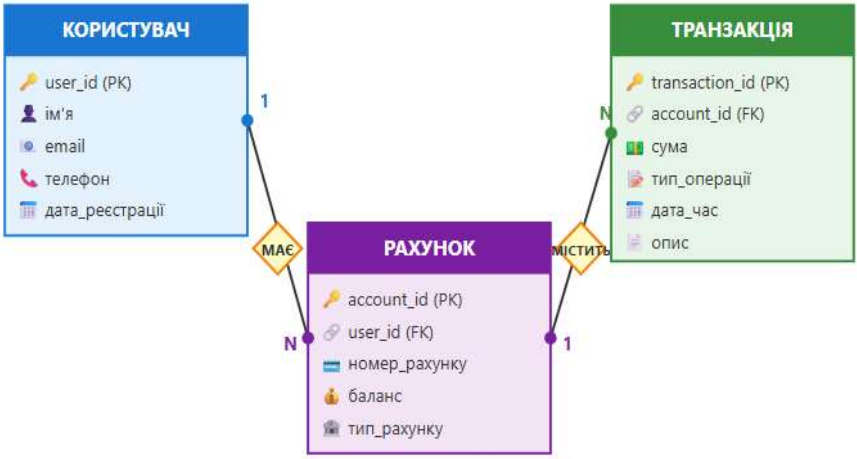

Рисунок 2.5 – Приклад спрощеної ER-діаграми (користувачі – рахунки – транзакції).

Проблеми етапу проєктування

- ✓ Надмірна деталізація, яка ускладнює розробку.
- ✓ Використання застарілих технологій (ризик при довгих проєктах).
- ✓ Відрив архітектури від реальних потреб користувачів.

2.2.2.3. Реалізація (кодинг)

Етап реалізації є центральним у каскадній моделі, адже саме тут

програмний продукт переходить від абстрактної архітектури та діаграм до реального коду (рис.2.6). Якщо попередні етапи відповідали на питання «що робити» і «як робити», то на етапі реалізації відбувається створення продукту засобами програмування.

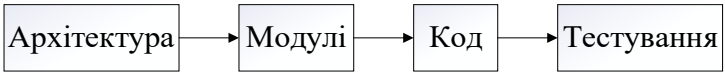

Рисунок 2.6 – Схема «Від проєктування до коду»

Основні завдання етапу:

1. Розробка модулів.

✓ Кожен модуль системи створюється окремо на основі технічних специфікацій.

✓ Кодування може виконуватися паралельно кількома командами.

2. Використання стандартів і практик.

✓ Дотримання внутрішніх та міжнародних стандартів кодування.

✓ Використання патернів проєктування для підвищення якості й повторного використання коду.

3. Інтеграція з інструментами.

✓ Використання систем контролю версій (Git) (рис.2.7).

✓ CI/CD-процеси (автоматичне збирання та тестування).

4. Внутрішнє тестування модулів.

✓ Юніт-тести, статичний аналіз коду.

✓ Перевірка кожного модуля перед інтеграцією в систему.

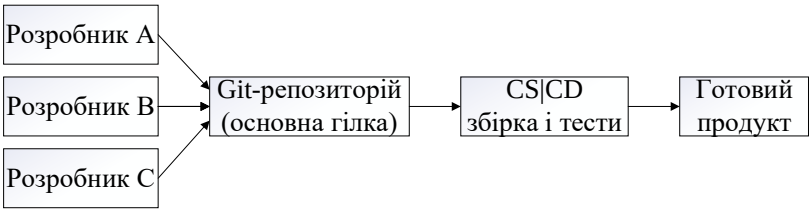

Рисунок 2.7 – Приклад організації командної роботи з Git

Артефакти етапу реалізації:

✓ Програмний код, що відповідає специфікації.

✓ Внутрішня документація (коментарі в коді, README, технічні описи).

✓ Базові тестові сценарії для перевірки модулів.

Приклад.

У випадку банківської системи реалізація передбачає:

✓ написання модуля авторизації на основі визначених алгоритмів шифрування;

✓ реалізацію платіжного модуля, який інтегрується з API процесингового центру;

✓ створення модуля формування звітів для користувачів.

Проблеми етапу реалізації

✓ Розрив між документацією та практикою. Програмісти можуть інтерпретувати проєктні документи по-різному.

✓ Людський фактор. Помилки в коді можуть коштувати дорого, особливо у великих системах.

✓ Складність інтеграції. Якщо модулі розробляються ізольовано, пізніше виникають проблеми при збиранні всієї системи.

2.2.2.4. Тестування

Фаза тестування в каскадній моделі є критично важливою, адже саме тут перевіряється, чи відповідає створений продукт вимогам, визначеним на першому етапі. Тестування дозволяє знайти помилки до того, як система буде передана замовнику, і мінімізувати ризик відмови в експлуатації.

Основні завдання тестування

1. Верифікація відповідності вимогам. Перевіряється, чи реалізовані всі функції, описані в SRS.

2. Виявлення дефектів. Знаходяться помилки у функціональності, інтерфейсі, продуктивності.

3. Оцінка якості. Перевірка продуктивності, безпеки, надійності, сумісності з різними платформами.

Типи тестування (рис.2.8):

✓ Модульне тестування (Unit testing). Перевіряються окремі функції та модулі.

✓ Інтеграційне тестування. Перевіряється взаємодія модулів між собою.

✓ Системне тестування. Перевіряється робота всієї системи як цілісного продукту.

✓ Приймальне тестування (Acceptance testing). Остаточна перевірка продукту замовником перед впровадженням.

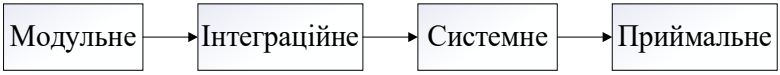

Рисунок 2.8 – Послідовність видів тестування

Артефакти етапу тестування:

✓ Тест-план – документ, що описує стратегію й обсяг тестування.

✓ Тест-кейси – детальні сценарії перевірки окремих функцій.

✓ Звіти про тестування – результати виконання тестів, знайдені дефекти.

Приклад.

Для банківської системи тестування включає:

✓ перевірку правильності авторизації з різними ролями користувачів;

✓ тестування швидкості обробки транзакцій при високому навантаженні;

✓ перевірку безпеки (наприклад, стійкість до SQL-ін’єкцій).

Проблеми етапу тестування:

✓ Висока вартість виправлення помилок (рис.2.9). Чим пізніше знайдена помилка, тим дорожче її виправити.

✓ Неповне тестування. Завжди є ризик, що не всі сценарії були перевірені.

✓ Обмежені терміни. На тестування часто виділяють менше часу, ніж потрібно.

Рисунок 2.9 – Графік «Вартість виправлення дефекту залежно від етапу» (чим пізніше знайдена помилка, тим дорожче).

2.2.2.5. Впровадження та супровід

Фаза впровадження – це момент істини для будь-якого ІТ-проєкту. Саме тут продукт стає доступним кінцевим користувачам і починає працювати в реальному середовищі. Після цього настає період супроводу, протягом якого система отримує оновлення, виправлення та підтримку (рис.2.10).

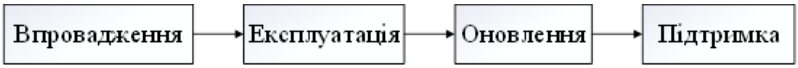

Рисунок 2.10 – «Життєвий цикл після впровадження»

Основні завдання впровадження:

1. Інсталяція продукту.
✓ Перенесення програмного забезпечення в робоче середовище.
✓ Налаштування серверів, баз даних, мережевої інфраструктури.
2. Навчання користувачів.
✓ Проведення тренінгів для персоналу.
✓ Розробка інструкцій і довідкових матеріалів.
3. Початок роботи системи.
✓ Пілотний запуск (обмежене коло користувачів).
✓ Повноцінний запуск (production release).

Основні завдання супроводу (рис.2.11):

1. Виправлення помилок. Навіть після тестування можуть виникати непередбачені дефекти.

2. Оновлення системи. Додавання нових функцій, адаптація до змін у бізнесі.

3. Технічна підтримка користувачів. Helpdesk, консультації, документація.

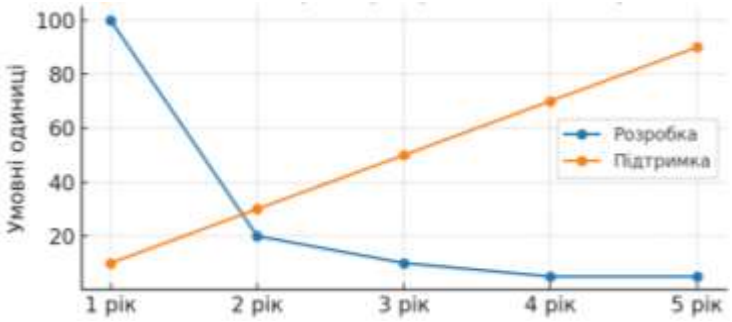

Рисунок 2.11 – Діаграма: витрати на підтримку та розробку

Артефакти етапу впровадження та супроводу:

- ✓ Інструкції з експлуатації.
- ✓ Керівництво користувача.
- ✓ Звіти про помилки та оновлення.

Приклад.

У банківській системі впровадження включає:

- ✓ розгортання сервера інтернет-банкінгу;
- ✓ міграцію баз даних із тестового середовища;
- ✓ навчання співробітників кол-центру;
- ✓ регулярні оновлення системи безпеки.

Проблеми етапу:

- ✓ Складність інтеграції з існуючими системами.
- ✓ Опір користувачів новим рішенням.
- ✓ Висока вартість технічного супроводу.

2.3. Стандарти PMBOK

PMBOK (Project Management Body of Knowledge) – це збірник стандартів, методів, процесів та найкращих практик управління проєктами, який розроблений і підтримується міжнародною організацією Project Management Institute (PMI).

2.3.1. Що таке PMBOK і роль Project Management Institute (PMI)

PMBOK не є конкретною методологією (як, наприклад, Waterfall чи Agile), а радше рамковим стандартом, який описує, що саме потрібно враховувати при управлінні проєктами, незалежно від галузі чи типу проєкту (рис.2.12).

Рисунок 2.12 – PMBOK – не методологія, а набір знань

Project Management Institute (PMI), Заснований у 1969 році в США, є найбільшою у світі професійною асоціацією з управління проєктами, яка видає міжнародно визнані сертифікації (рис.2.13):

- ✓ PMP (Project Management Professional) – «золотий стандарт» у сфері управління проєктами.
- ✓ CAPM, PgMP, PMI-ACP та інші.

Рисунок 2.13 – Хронологія PMI

PMI розробляє та регулярно оновлює PMBOK, роблячи його універсальним інструментом для проєктних менеджерів у будь-якій сфері.

Роль PMBOK у проєктному менеджменті:

✓ Єдиний словник термінів, який дозволяє всім учасникам проєкту «говорити однією мовою».

✓ Набір найкращих практик. Базується на досвіді тисяч проєктів у різних країнах і сферах.

✓ Гнучкість. PMBOK не прив'язує до однієї моделі (каскадної чи гнучкої), а може використовуватися в будь-якому контексті.

✓ Міжнародний стандарт. Використовується в бізнесі, IT, будівництві, авіації, обороні, державних проєктах.

Важливість для IT-проєктів:

✓ Забезпечує структуроване управління навіть у складних програмах.

✓ Дає підхід до роботи з ризиками, змінами, ресурсами, що особливо важливо у великих командах.

✓ Є базою, яку часто комбінують з Agile/Scrum (гібридний підхід).

2.3.2. П'ять процесних груп PMBOK

2.3.2.1. Ініціація

Етап ініціації – це старт проєкту, коли формулюється його мета, визначаються ключові учасники та ухвалюється рішення, чи варто взагалі розпочинати проєкт (рис.2.14). Це фундамент, від якого залежить подальший успіх.

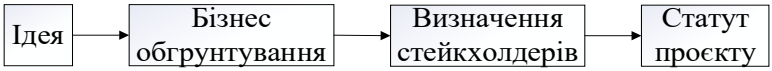

Рисунок 2.14 – Процес ініціації»

Ініціація відповідає на питання:

✓ Чому ми запускаємо цей проєкт?

✓ Яку цінність він створить?

✓ Хто буде залучений?

✓ Які обмеження існують (бюджет, час, ресурси)?

Основні завдання етапу:

1. Формування бізнес-обґрунтування. Чітко описати, яку проблему вирішує проєкт і яку вигоду він приносить.

2. Визначення зацікавлених сторін (stakeholders). Власники бізнесу, користувачі, розробники, регулятори.

3. Призначення керівника проєкту (Project Manager). Людина, яка відповідає за планування й управління.

4. Створення статуту проєкту (Project Charter). Документ, що офіційно санкціонує проєкт і визначає його межі.

Артефакти етапу:

✓ Project Charter (Статут проєкту). Містить цілі, очікувані результати, бюджет, терміни, відповідальних осіб (рис.2.15).

✓ Реєстр зацікавлених сторін. Список ключових учасників із зазначенням їхніх ролей та впливу.

Елемент	Зміст
Цілі	Запуск мобільного банкінгу
Бюджет	$2 млн
Терміни	18 місяців
Відповідальні	Project Manager, керівництво банку

Рисунок 2.15 – Приклад структури Project Charter

Приклад.

У випадку ІТ-проєкту для банку ініціація може включати:

✓ обґрунтування запуску мобільного банкінгу для підвищення клієнтського досвіду;

✓ визначення основних стейкхолдерів: керівництво банку, відділ ІТ, клієнти, регулятор;

✓ створення статуту, де визначено бюджет у $2 млн, термін реалізації 18 місяців.

Проблеми на етапі ініціації:

✓ Нечітко сформульовані цілі → призводять до провалів у

майбутньому.

- ✓ Відсутність згоди між стейкхолдерами.
- ✓ Занадто оптимістичні оцінки бюджету чи термінів.

2.3.2.2. Планування

Етап планування – це деталізація того, як буде реалізовано проєкт. Він забезпечує основу для контролю та управління, дозволяючи координувати команду й ресурси.

Якщо на етапі ініціації відповіли на питання «Чому ми робимо проєкт?», то на етапі планування відповідають на питання:

- ✓ Що саме потрібно зробити?
- ✓ Як і коли це буде зроблено?
- ✓ Які ресурси потрібні?
- ✓ Які ризики існують і як ними керувати?

Основні завдання планування:

1. Формування плану управління проєктом. Головний документ, який включає всі допоміжні плани (ресурсів, ризиків, комунікацій).

2. Розробка структури декомпозиції робіт (WBS – Work Breakdown Structure). Поділ проєкту на зрозумілі завдання й підзавдання (рис.2.16).

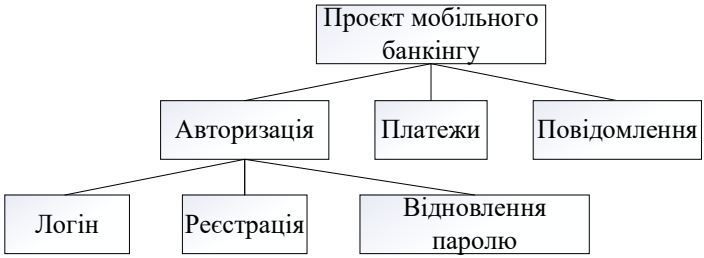

Рисунок 2.16 – Схема WBS (дерево завдань).

3. Формування календарного плану (графік). Використання діаграм Ганта, мережевих графіків (CPM, PERT).

4. Оцінка бюджету та ресурсів. Витрати на людей, обладнання, ліцензії, ризики.

5. Планування комунікацій і взаємодії зі стейкхолдерами.

Артефакти етапу:

✓ Project Management Plan (план управління проєктом).

✓ WBS (Work Breakdown Structure).

✓ Графік робіт (діаграма Ганта (рис.2.17, мережевий граф).

✓ План управління ризиками.

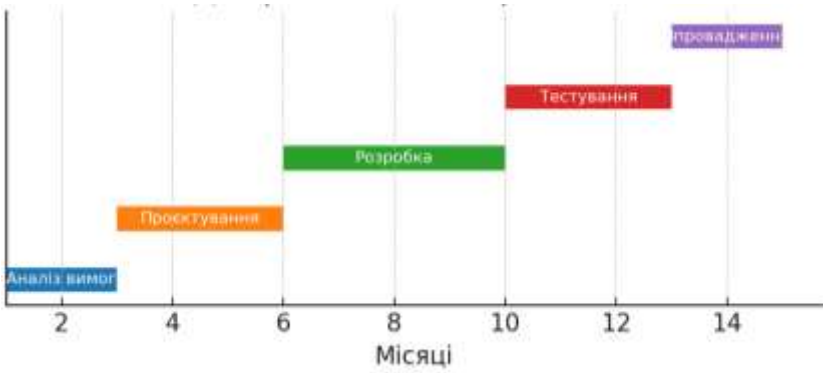

Рисунок 2.17 – Приклад діаграми Ганта.

Приклад

У проєкті створення мобільного банкінгу планування може включати:

✓ побудову WBS: модуль авторизації, модуль платежів, модуль повідомлень;

✓ складання діаграми Ганта на 18 місяців;

✓ бюджет – $2 млн, включаючи зарплати команди, сервери, ліцензії;

✓ визначення ризиків: затримки з інтеграцією, зміни регуляторних вимог.

Проблеми на етапі планування

✓ Занадто оптимістичні оцінки термінів і вартості.

✓ Відсутність узгодження планів між різними командами.

✓ Ігнорування ризиків або недооцінка їхнього впливу.

2.3.2.3. Виконання

Етап виконання – це реалізація запланованих робіт. Тут команда втілює всі напрацювання з етапу планування: створює продукт, надає послуги, координує ресурси та комунікує зі стейкхолдерами (рис.2.18).

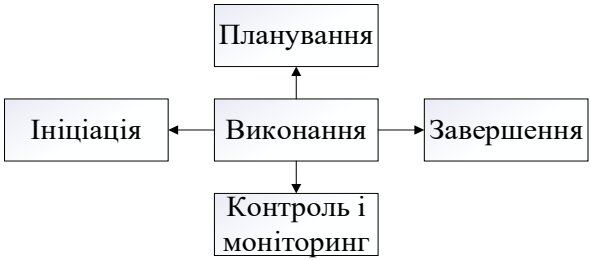

Рисунок 2.18 – Виконання, як центр життєвого циклу

Якщо на попередніх етапах відповідали на питання «Чому?» і «Як?», то на цьому етапі відповідають на питання:

- ✓ Що конкретно робимо зараз?
- ✓ Хто виконує завдання?
- ✓ Як організована взаємодія між учасниками?

Основні завдання виконання (рис.2.19):

1. Розподіл завдань і управління командою.
- ✓ Виконання WBS у реальних умовах.
- ✓ Мотивація та управління динамікою в команді.
2. Забезпечення якості продукту.
- ✓ Виконання планів тестування, код-рев'ю, аудитів.
3. Комунікація зі стейкхолдерами.
- ✓ Регулярні зустрічі, звіти про прогрес, демо-версії.
4. Закупівлі та управління постачальниками.
- ✓ Контракти, ліцензії, обладнання.
5. Управління змінами.
- ✓ Реакція на нові вимоги, їх офіційне документування.

Артефакти етапу:

- ✓ Звіти про виконання (progress reports).

- ✓ Проміжні версії продукту.
- ✓ Протоколи зустрічей.
- ✓ Документація про зміни.

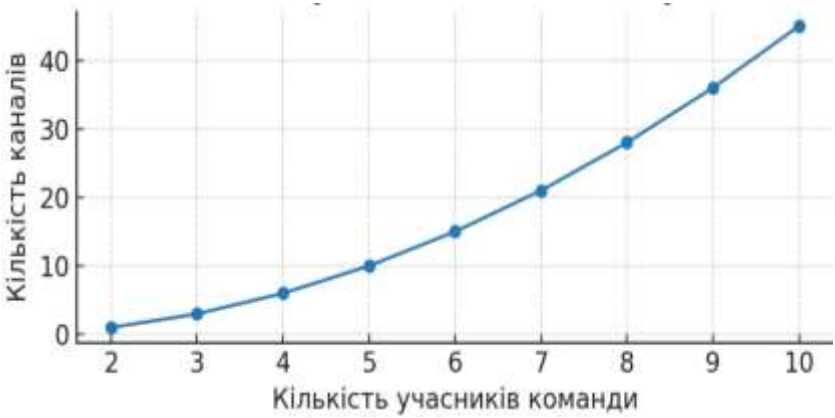

Рисунок 2.19 – Діаграма «Комунікаційні канали в команді» (залежність кількості каналів від кількості учасників: формула n(n-1)/2).

Приклад.

У проєкті мобільного банкінгу:

- ✓ команда розробників створює модулі (авторизація, платежі, повідомлення);
- ✓ тестувальники перевіряють якість;
- ✓ Project Manager готує щотижневі звіти для керівництва банку;
- ✓ закуповується серверне обладнання.

Проблеми на етапі виконання:

- ✓ Перевантаження ресурсів (занадто багато завдань на обмежену команду).
- ✓ Конфлікти в команді через комунікацію.
- ✓ Низька мотивація.
- ✓ Виникнення непередбачених змін (регуляторні вимоги, нові функції).

2.3.2.4. Моніторинг і контроль

Це постійне відстеження прогресу проєкту, перевірка, чи відповідає виконання планам, та внесення коригувальних дій у разі відхилень (рис.2.20).

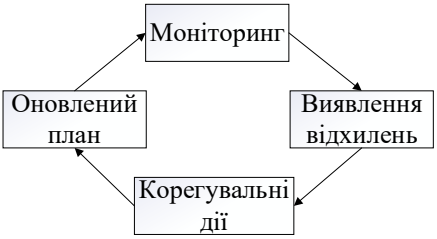

Рисунок 2.20 – Цикл Моніторингу

Основна мета – не допустити виходу проєкту за межі бюджету, термінів і якості.

Відповідає на питання:

- ✓ Чи йдемо ми за планом?
- ✓ Які відхилення вже є?
- ✓ Що потрібно змінити, щоб досягти мети?

Основні завдання:

1. Вимірювання прогресу. Використання ключових показників (KPI, EVM – Earned Value Management) (рис.2.21).
2. Виявлення відхилень. Порівняння фактичних результатів із планом.
3. Прийняття коригувальних дій. Перепланування, зміна розподілу ресурсів.
4. Контроль якості. Перевірки, рев'ю, тестування.
5. Управління ризиками. Моніторинг і реагування на нові ризики.

Артефакти:

- ✓ Регулярні звіти про статус.
- ✓ KPI-дашборди.
- ✓ Оновлений план управління проєктом.
- ✓ Документація про ризики.

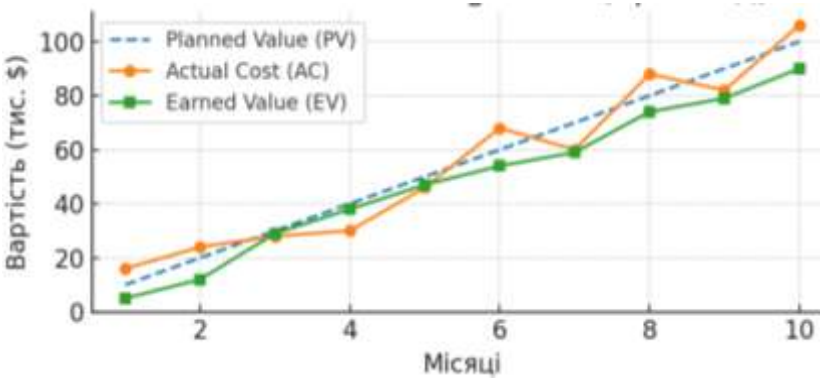

Рисунок 2.21 – Приклад графіку Earned Value
(Planned Value vs Actual Cost vs Earned Value).

Приклад.

У мобільному банкінгу:

- ✓ щотижневі звіти показують затримку в модулі «Платежі» на 2 тижні;
- ✓ менеджер ухвалює рішення збільшити команду тестувальників;
- ✓ проводиться контроль якості через додаткові рев'ю.

Проблеми:

- ✓ Недостатня прозорість (немає зрозумілих метрик).
- ✓ Несвоєчасне реагування на відхилення.
- ✓ Надмірна бюрократія (зайва звітність).

2.3.2.5. Завершення

Етап завершення – це офіційне закриття проєкту або його фази. Він включає передачу кінцевого продукту замовнику, підбиття підсумків та формування уроків для майбутніх проєктів (рис.2.22).

Відповідає на питання:

- ✓ Чи досягли ми запланованих цілей?
- ✓ Які уроки можна винести для наступних проєктів?
- ✓ Чи задоволений замовник і стейкхолдери?

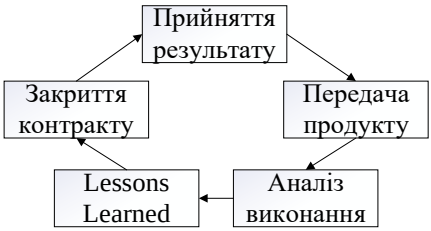

Рисунок 2.22 – Етап завершення

Основні завдання (рис.2.23):

1. Формальне прийняття результату. Замовник офіційно підтверджує, що продукт відповідає вимогам.

2. Передача продукту в експлуатацію. Технічна документація, навчання користувачів.

3. Аналіз виконання. Порівняння фактичних результатів із планом (час, бюджет, якість).

4. Уроки, винесені з проєкту (lessons learned). Збір досвіду команди, рекомендації для майбутніх проєктів.

5. Закриття контрактів. Завершення співпраці з підрядниками, оплата рахунків.

Check-list завершення проєкту

- ✦ Отримано офіційне підтвердження від замовника
- ✦ Передано продукт та документацію
- ✦ Проведено навчання користувачів
- ✦ Складено фінальний звіт
- ✦ Задокументовано lessons learned
- ✦ Закрито контракти та рахунки

Рисунок 2.23 – Check-list «Що має бути зроблено для закриття проєкту»

Артефакти:

Фінальний звіт про проєкт.

✓ Lessons Learned Report.

✓ Документи про передачу продукту замовнику.

✓ Акт закриття контрактів.

Приклад.

У банківському проєкті:

✓ продукт (мобільний додаток) передано клієнту;

✓ підготовлено документацію та проведено навчання працівників;

✓ звіт показав виконання проєкту з перевищенням бюджету на 5 %, але з дотриманням термінів;

✓ у звіті Lessons Learned зафіксовано необхідність раніше залучати регулятора.

Проблеми:

✓ Замовник не приймає продукт через невідповідність очікуванням.

✓ Відсутність належного аналізу досвіду (втрата уроків).

✓ Неповне закриття контрактів → юридичні ризики.

2.3.3. Десять областей знань PMBOK

2.3.3.1. Управління інтеграцією (Project Integration Management)

Управління інтеграцією – це узгодження всіх процесів і завдань проєкту в єдину систему (рис.2.24).

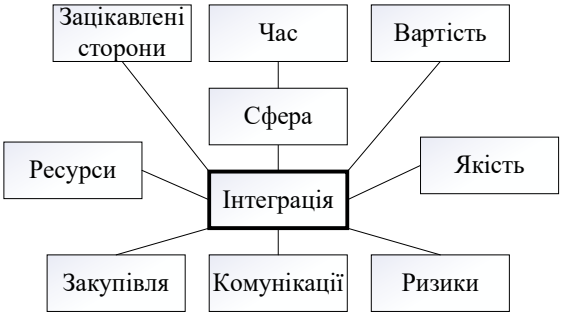

Рисунок 2.24 – Інтеграція як центр, що пов’язує інші області знань (час, бюджет, якість, ризики тощо).

Це область знань, яка об’єднує результати роботи з інших напрямів

(час, бюджет, ризики, комунікації тощо) і гарантує, що всі частини проєкту працюють на досягнення єдиної мети.

Основні процеси управління інтеграцією (згідно PMBOK) (рис.2.25):

1. Розробка Статуту проєкту (Project Charter). Формальне схвалення проєкту керівництвом.

2. Розробка плану управління проєктом. Комплексний документ, що визначає, як саме буде виконуватися і контролюватися проєкт.

3. Керування виконанням проєкту. Забезпечення узгодженої роботи всіх учасників.

4. Моніторинг і контроль інтеграції. Координація змін, управління взаємозалежностями.

5. Завершення проєкту або фази. Формальне закриття робіт, узагальнення результатів.

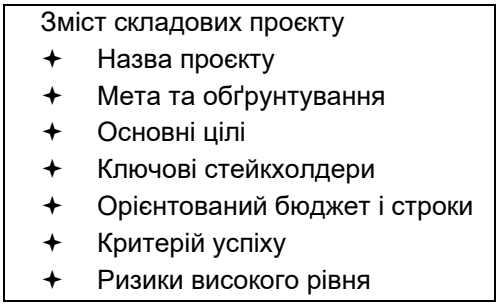

Рисунок 2.25 – Приклад структури Project Charter з основними блоками (мета, бюджет, строки, стейкхолдери).

Роль у проєкті:

- ✓ Узгоджує різні підходи (Agile, Waterfall, гібридні).
- ✓ Забезпечує єдине інформаційне поле для команди та замовників.
- ✓ Допомагає уникати «розривів» між підсистемами проєкту.

Приклад.

У проєкті створення мобільного банкінгу інтеграція передбачає:

- ✓ поєднання роботи команд розробників і тестувальників;
- ✓ координацію між банком, підрядниками та регуляторами;

✓ управління змінами, коли клієнт хоче додати нові функції (наприклад, «миттєві платежі»).

Проблеми без інтеграції:

✓ Різні команди працюють у «силах», без єдиного плану.

✓ Конфлікти між цілями (ІТ хоче швидкість, бізнес – безпеку, замовник – мінімальний бюджет).

✓ Подвійна робота і дублювання завдань.

2.3.3.2. Управління змістом (Scope Management)

Управління змістом (scope) – це визначення та контроль того, що входить і не входить у проєкт.

Це критична область знань, оскільки більшість проблем проєктів виникає через «розповзання змісту» (scope creep) – коли додаються нові завдання без корекції бюджету та термінів.

Основні процеси управління змістом:

1. Планування управління змістом. Розробка плану, як буде визначатися і контролюватися зміст.

2. Збір вимог. Виявлення потреб замовників та стейкхолдерів.

3. Визначення змісту. Формування детального опису продукту і проєкту.

4. Створення WBS (Work Breakdown Structure). Ієрархічна декомпозиція проєкту на роботи (2.26).

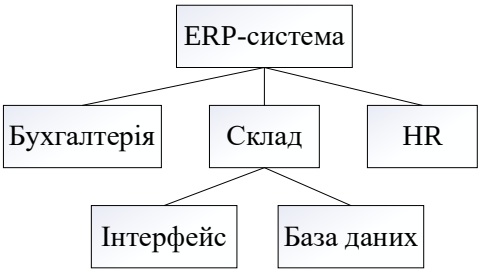

Рисунок 2.26 – Приклад WBS (ієрархічне дерево робіт).

5. Підтвердження змісту. Формальне затвердження виконаних робіт

замовником.

6. Контроль змісту. Відстеження змін та управління ними.

Артефакти:

✓ План управління змістом.

✓ Реєстр вимог.

✓ WBS та словник WBS.

✓ Документи про прийняття робіт.

Приклад.

У проєкті створення ERP-системи:

✓ на етапі збору вимог визначено модулі «Бухгалтерія», «Склад», «HR»;

✓ WBS деталізував роботи до рівня «створення інтерфейсу складу», «налаштування бази даних»;

✓ додавання нової функції «аналітика продажів» стало предметом офіційної процедури зміни, а не «просто ще одне завдання».

Проблеми без управління змістом:

✓ «Scope creep» – неконтрольоване розширення робіт (без узгодження змін бюджету/терминів) (рис.2.27).

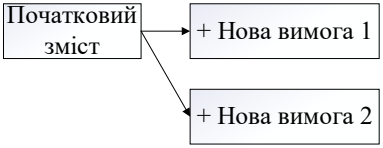

Рисунок 2.27 – «Scope creep» (як нові вимоги розширюють зміст).

✓ Втрата контролю над термінами та бюджетом.

✓ Конфлікти з замовником щодо очікуваних результатів.

2.3.3.3. Управління часом (Schedule Management)

Управління часом – це процеси, які забезпечують своєчасне завершення проєкту.

Воно включає планування, оцінку тривалості завдань, складання графіка і контроль за його дотриманням.

Основні процеси:

1. Планування управління розкладом. Визначення методології та інструментів для побудови графіка (MS Project, Jira, Gantt).
2. Визначення діяльностей. Розбиття WBS на конкретні завдання.
3. Встановлення послідовності робіт. Визначення залежностей між завданнями (FS, SS, FF, SF).
4. Оцінка тривалості. Методи: експертні оцінки, аналоги, PERT-аналіз.
5. Розробка розкладу. Створення діаграми Ганта, мережевих діаграм (CPM, PERT).
6. Контроль розкладу. Відстеження прогресу, виявлення відхилень.

Артефакти:

- ✓ План управління розкладом.
- ✓ Мережевий графік.
- ✓ Діаграма Ганта (рис.2.28).
- ✓ Критичний шлях (Critical Path) (рис.2.29).

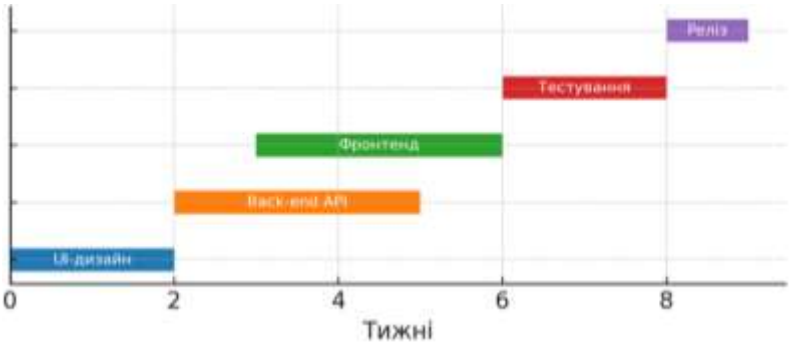

Рисунок 2.28 – Приклад діаграми Ганта (умовний).

Приклад.

У проєкті розробки мобільного застосунку:

- ✓ WBS розбитий на «UI-дизайн», «Back-end API», «Тестування»;
- ✓ залежності: тестування починається після завершення розробки;
- ✓ критичний шлях показав, що затримка у дизайні на 2 тижні змістить реліз.

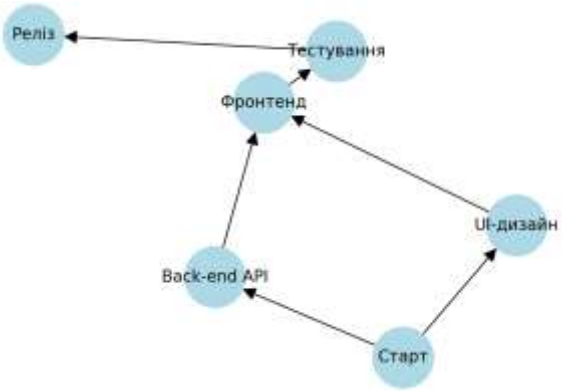

Рисунок 2.29 – Мережева діаграма з критичним шляхом.

Проблеми без управління часом:

- ✓ Нереалістичні строки.
- ✓ «Вузькі місця» в процесі, які не враховані в плані.
- ✓ Затримки, що накопичуються і ведуть до зриву дедлайнів.

2.3.3.4. Управління вартістю (Cost Management)

Управління вартістю – це процеси планування, оцінки, формування бюджету та контролю витрат, щоб проєкт завершився у межах затвердженого бюджету (рис.2.30).

Рисунок 2.30 – Приклад структури бюджету (діаграма).

Основні процеси:

1. Планування управління вартістю. Визначення, як будуть оцінюватися, бюджетуватися і контролюватися витрати.

2. Оцінка вартості. Визначення приблизних ресурсів і їхньої вартості (експертна оцінка, параметричні моделі, аналоги).

3. Формування бюджету. Складання кошторису та розподіл бюджету по фазах/завданнях.

4. Контроль вартості. Відстеження фактичних витрат, аналіз відхилень, використання метрик Earned Value (рис.2.31).

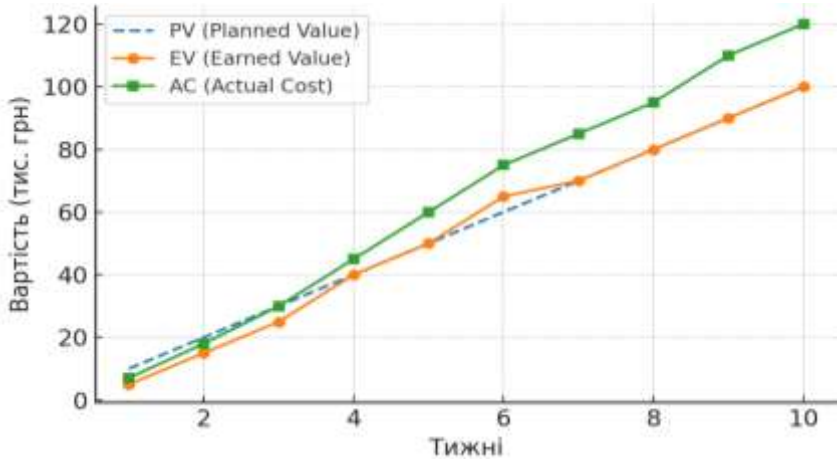

Рисунок 2.31 – Графік Earned Value (PV, EV, AC).

Артефакти:

- ✓ План управління вартістю.
- ✓ Кошторис (Cost Estimate).
- ✓ Бюджет проєкту (Cost Baseline).
- ✓ Звіти Earned Value (CPI, SPI).

Приклад.

У проєкті створення вебпорталу для банку:

- ✓ оцінка показала витрати 500 тис. грн на розробку;

✓ бюджет був розподілений: 40 % на розробку, 20 % на тестування, 30 % на інтеграцію, 10 % на резерв;

✓ у процесі виконання аналіз CPI = 0.9 показав перевитрати ресурсів.

Проблеми без управління вартістю:

✓ Перевищення бюджету.

✓ Відсутність фінансового контролю.

✓ Неможливість обґрунтувати додаткове фінансування.

2.3.3.5. Управління якістю (Quality Management)

Управління якістю – це процеси, що гарантують, що продукт і сам проєкт відповідають вимогам замовника та стандартам (рис.2.32).

Якість – це не тільки відсутність дефектів, а й відповідність очікуванням та задоволення стейкхолдерів.

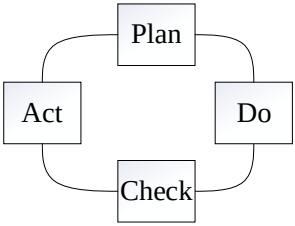

Рисунок 2.32 – Цикл управління якістю (PDCA = Plan – Do – Check – Act).

Основні процеси:

1. Планування якості. Визначення стандартів, методів перевірки, метрик (наприклад, ISO 9001, CMMI).

2. Забезпечення якості. Впровадження процесів, що гарантують якість (code review, CI/CD, автоматизоване тестування).

3. Контроль якості. Перевірка результатів: тести, аудит, інспекції.

Артефакти:

✓ План управління якістю.

✓ Метрики якості (Defect Density, Code Coverage) (рис.2.33).

✓ Чек-листи перевірок.

✓ Звіти з тестування.

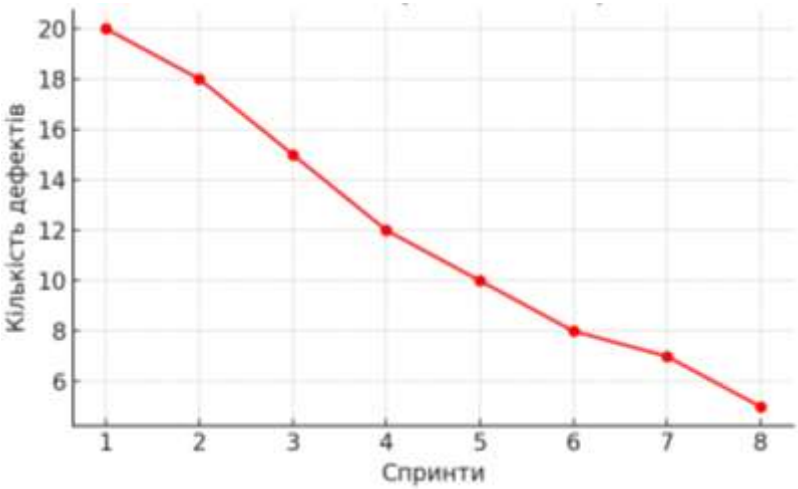

Рисунок 2.33 – Приклад діаграми контролю якості (Defects per Sprint).

Приклад

У проєкті створення онлайн-магазину:

✓ на етапі планування визначили ціль: <2 % критичних дефектів на продакшн;

✓ забезпечення якості: застосування автоматизованих тестів і code review;

✓ контроль: регулярне проведення юзабіліті-тестів і аудитів безпеки.

Проблеми без управління якістю:

✓ Зростання кількості дефектів.

✓ Втрата довіри клієнтів.

✓ Переробки, що ведуть до збільшення витрат і затримок.

2.3.3.6. Управління ресурсами (Resource Management)

Управління ресурсами включає процеси, пов’язані з людськими, матеріальними та фінансовими ресурсами, які потрібні для виконання проєкту.

Основна мета – оптимальне використання ресурсів для досягнення цілей.

Основні процеси:

1. Планування ресурсів. Визначення типів і кількості ресурсів (людей, обладнання, матеріалів).

2. Оцінка ресурсів. Розрахунок необхідних обсягів ресурсів для завдань WBS.

3. Залучення команди. Найм, формування та введення нових членів команди.

4. Розвиток команди. Навчання, тімбілдинг, підвищення мотивації.

5. Управління командою. Моніторинг ефективності, вирішення конфліктів.

6. Контроль використання ресурсів. Відстеження витрат ресурсів, балансування навантаження.

Артефакти:

- ✓ Матриця ресурсів (Resource Histogram) (рис.2.34).
- ✓ RACI-матриця (розподіл ролей і відповідальності) (рис.2.35).
- ✓ План розвитку команди.

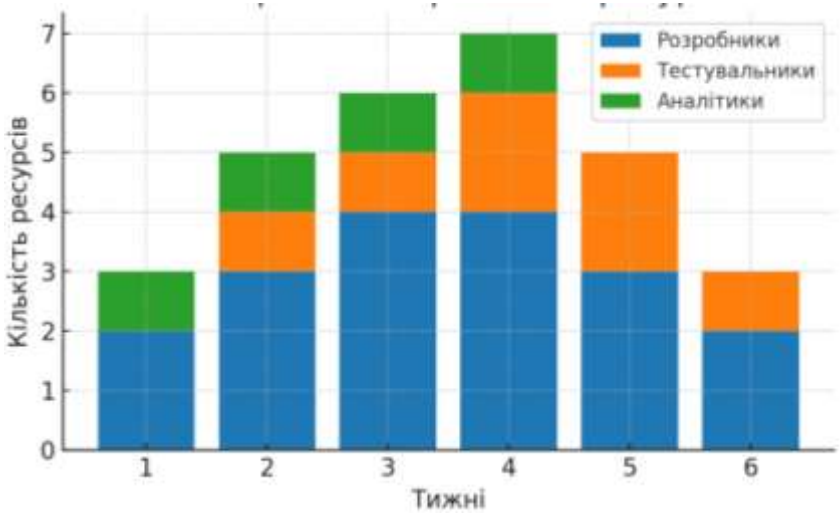

Рисунок 2.34 – Гістограма ресурсів (розподіл завантаження по тижнях).

Завдання	PM	Аналітик	Розробник	Тестувальник	DevOps
Аналіз вимог	A	R	I	I	I
Розробка UI	R	C	R	I	I
Back-end API	C	I	R	C	I
Тестування	C	I	C	R	C
Реліз	I	I	I	I	R

Рисунок 2.35 – Приклад RACI-матриця (таблиця ролей).

Приклад.

У проєкті створення мобільного банкінгу:

✓ сформували RACI-матрицю для визначення відповідальності;

✓ використали гістограму ресурсів для розподілу завантаження розробників;

✓ команда проходила регулярні тренінги з кібербезпеки.

Проблеми без управління ресурсами:

✓ Перевантаження окремих людей.

✓ Конфлікти через нечіткий розподіл ролей.

✓ Низька мотивація та плинність кадрів.

2.3.3.7. Управління комунікаціями (Communications Management)

Управління комунікаціями охоплює процеси, які забезпечують своєчасний збір, створення, розповсюдження та збереження інформації в межах проєкту (рис.2.36).

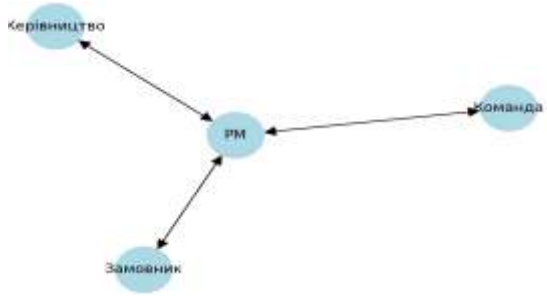

Рисунок 2.36 – Схема інформаційних потоків у проєкті.

Дослідження PMI показують, що понад 50 % проблем у проєктах виникають через неякісну комунікацію.

Основні процеси:

1. Планування комунікацій.

✓ Визначення каналів, форматів і частоти обміну інформацією (рис.2.37).

✓ Приклад: щотижневі стендапи, звіти у Confluence, щомісячні зустрічі зі стейкхолдерами.

2. Забезпечення комунікацій. Проведення мітингів, ведення документації, оновлення дашбордів.

3. Моніторинг комунікацій. Перевірка, чи доходить інформація до всіх учасників і чи зрозуміла вона.

Стейкхолдери	Канал	Частота	Формат
PM	Stack / Email	Щодня	Текст
Команда розробки	Jira / Stand-up	Щодня	Мітинг + таски
Замовник	Звіти / Презентації	Щотижня	PDF/Слайди
Керівництво	Щомісячні огляди	Щомісяця	Зустріч + Звіт

Рисунок 2.37 – Матриця комунікацій (стейкхолдер – канал – частота – формат).

Артефакти:

✓ План комунікацій.

✓ Канали зв'язку (Slack, Teams, email).

✓ Звіти про статус проєкту.

✓ Дашборди Jira/Confluence.

Приклад.

У проєкті розробки ERP-системи:

✓ було визначено: щоденні стендапи (15 хв), щотижневі демо, щомісячний звіт у вигляді презентації;

✓ використання Jira та Confluence забезпечувало прозорість інформації;

✓ контроль комунікацій виявив, що частина стейкхолдерів потребує

резюме англійською.

Проблеми без управління комунікаціями:

- ✓ Інформаційний хаос.
- ✓ Подвійні трактування завдань.
- ✓ Конфлікти між замовником і командою.

2.3.3.8. Управління ризиками (Risk Management)

Управління ризиками – це систематичний процес виявлення, аналізу та реагування на ризики, які можуть вплинути на проєкт.

Мета – мінімізувати ймовірність негативних подій та максимально використати можливості.

Основні процеси:

1. Планування управління ризиками. Визначення методів і підходів до роботи з ризиками.
2. Ідентифікація ризиків. \

- ✓ Складання реєстру ризиків (Risk Register).
- ✓ Використання інтерв'ю, мозкового штурму, SWOT-аналізу.

3. Кількісний і якісний аналіз.

- ✓ Якісний: оцінка ймовірності та впливу (Risk Matrix).
- ✓ Кількісний: моделювання (Monte Carlo, EMV).

4. Планування реагування на ризики. - Уникнення, передача, пом'якшення, прийняття.
5. Моніторинг і контроль ризиків. -Регулярний перегляд реєстру, оновлення планів реагування.

Артефакти:

- ✓ План управління ризиками.
- ✓ Матриця ризиків (Impact/Probability) (рис.2.38).

Ймовірність/Вплив	Низький	Середній	Високий
Низька			
Середня			
Висока			Критична зона

Рисунок 2.38 – Матриця ризиків (Probability vs Impact).

✓ Реєстр ризиків (табл.2.3).

Таблиця 2.3 – Приклад реєстру ризиків

ID	Опис ризику	Ймовірність	Вплив	Стратегія
R1	Затримка фінансування	Висока	Критична	Пом'якшення
R2	Відмова сервесів	Середня	Високий	Уникнення
R3	Звільнення ключового розробника	Середня	Середній	Прийняття
R4	Зміни вимог замовника	Висока	Високий	Пом'якшення

✓ Звіти з моніторингу.

Приклад.

У проєкті державної ІТ-системи:

✓ було виявлено ризик затримки фінансування (ймовірність висока, вплив критичний);

✓ обрали стратегію пом'якшення: передбачили резерв у 15 % бюджету;

✓ під час реалізації ризик справдився, але завдяки резерву проєкт був завершений.

Проблеми без управління ризиками:

✓ Неочікувані кризи.

✓ Переробки, перевищення бюджету і затримки.

✓ Втрата довіри замовника.

2.3.3.9. Управління постачаннями (Procurement Management)

Управління постачаннями охоплює процеси, необхідні для закупівлі товарів, робіт і послуг у зовнішніх організацій.

У сучасних ІТ-проєктах це можуть бути:

✓ хмарні сервіси (AWS, Azure, GCP),

✓ обладнання (сервери, мережеві пристрої),

✓ аутсорсингові послуги (QA, дизайн, консалтинг).

Основні процеси (рис.2.39):

1. Планування постачань.

✓ Визначення, що робиться власними силами, а що закуповується

(рис.2.40).

- ✓ Підготовка плану закупівель.

2. Проведення закупівель.

- ✓ Запити комерційних пропозицій (RFP, RFQ).
- ✓ Оцінка постачальників, переговори, вибір.

3. Контроль контрактів.

- ✓ Відстеження виконання зобов'язань постачальниками.
- ✓ Управління змінами контракту.

4. Закриття контрактів.

- ✓ Перевірка поставок, фінальне приймання та закриття документів.

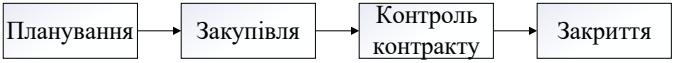

Рисунок 2.39 – Життєвий цикл контракту (Procurement Lifecycle).

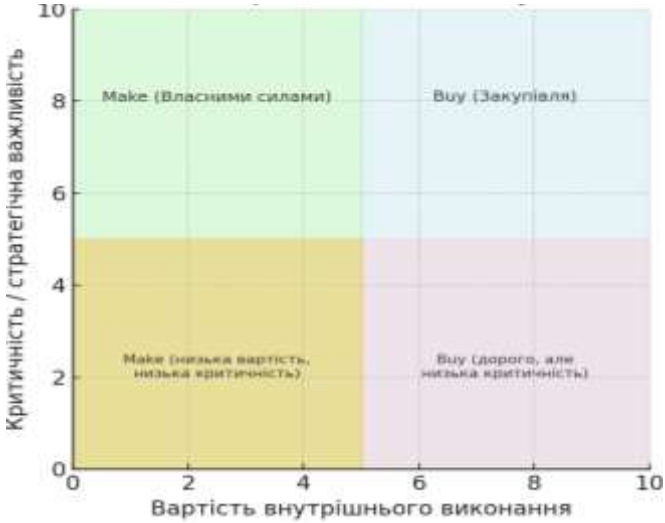

Рисунок 2.40 – Матриця "Make or Buy" (що робимо самі, що закуповуємо).

Артефакти:

- ✓ План постачань.

- ✓ Контракт / SLA.
- ✓ Рейтинг постачальників.
- ✓ Звіти з виконання контрактів.

Приклад.

У проєкті створення державної електронної платформи:

- ✓ частину робіт виконували внутрішні розробники;
- ✓ для забезпечення кібербезпеки було залучено зовнішній консалтинговий центр;
- ✓ контроль контракту виявив затримку постачання обладнання, що потребувало пролонгації договору.

Проблеми без управління постачаннями:

- ✓ Невчасні або неякісні поставки.
- ✓ Юридичні та фінансові ризики.
- ✓ Конфлікти з підрядниками.

2.3.3.10. Управління зацікавленими сторонами (Stakeholder Management)

Зацікавлені сторони (stakeholders) – це всі особи та організації, які прямо чи опосередковано впливають на проєкт або відчувають на собі його вплив.

Це можуть бути:

- ✓ замовник,
- ✓ кінцеві користувачі,
- ✓ керівництво компанії,
- ✓ команда розробки,
- ✓ державні органи, інвестори тощо.

Мета управління зацікавленими сторонами – визначити, проаналізувати та ефективно взаємодіяти з ними, щоб знизити конфлікти і підвищити шанси успіху проєкту.

Основні процеси:

1. Ідентифікація стейкхолдерів.

- ✓ Хто має інтерес у проєкті? Хто може впливати?
- ✓ Результат: реєстр стейкхолдерів.

2. Планування взаємодії.

- ✓ Визначення підходів до комунікацій.
- ✓ Використання матриці Power-Interest (рис.2.41).

3. Залучення стейкхолдерів.

- ✓ Регулярна участь у мітингах, демо, рев'ю.
- ✓ Прозорі звіти й активна комунікація.

4. Моніторинг залучення.

- ✓ Аналіз зворотного зв'язку.
- ✓ Корекція плану взаємодії.

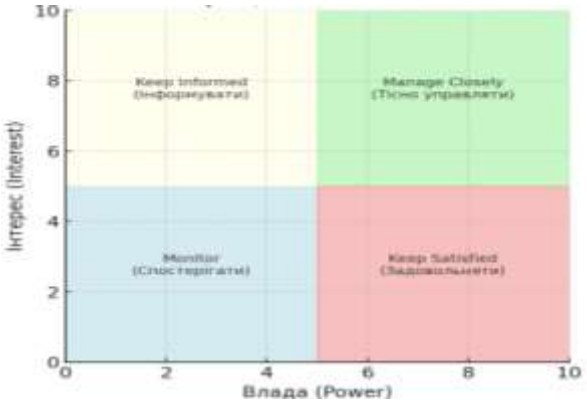

Рисунок 2.41 – Матриця Power-Interest (Влада-Інтерес)

Артефакти:

- ✓ Реєстр стейкхолдерів (табл.2.4)

Таблиця 2.4 – Приклад реєстру стейкхолдерів

ID	Стейкхолдери	Влада	Інтерес	Стратегія
S1	Керівництво банку	Висока	Високий	Manager Closery
S2	IT-департамент	Середня	Високий	Keep Informed
S3	Відділ продажів	Середня	Середній	Keep Satisfied
S4	Кінцеві користувачі	Висока	Високий	Keep Informed

- ✓ Матриця Power-Interest.

✓ План взаємодії зі стейкхолдерами.

✓ Звіти з моніторингу залученості.

Приклад.

У проєкті впровадження CRM-системи для банку:

✓ стейкхолдерами виступали керівництво банку, відділ продажів, IT-департамент і кінцеві користувачі;

✓ матриця Power-Interest показала, що керівництво має високу владу та високий інтерес → потребувало щотижневих звітів;

✓ користувачі мали високий інтерес, але низьку владу → отримували навчання та брали участь у тестуванні.

Проблеми без управління стейкхолдерами:

✓ Ігнорування інтересів ключових груп.

✓ Опір змінам і саботаж.

✓ Втрата підтримки керівництва.

2.3.4. Приклади застосування PMBOK у великих IT-програмних проєктах

PMBOK особливо корисний у великих і складних проєктах, де:

✓ задіяні сотні учасників;

✓ є численні зацікавлені сторони (державні органи, інвестори, підрядники);

✓ потрібно суворе документування, управління змінами, контроль бюджету та термінів.

Приклад 1. ERP-система для міжнародної корпорації

Контекст: корпорація з 50+ філіями в різних країнах вирішила впровадити уніфіковану ERP-систему, охоплює фінанси, логістику, HR і виробництво.

PMBOK: застосовані групи процесів – ініціація (обґрунтування ROI), планування (розбиття на хвилі релізів), моніторинг (KPI за якістю та вартістю).

Як застосували PMBOK:

✓ Інтеграційне управління – створено програмний офіс (PMO), що координував усі команди.

✓ Управління змістом – вимоги детально зафіксовані у «Scope Statement», щоб уникнути неконтрольованих змін.

✓ Управління ризиками – ідентифікували ризик невідповідності локального законодавства в різних країнах; створили план локалізації.

Результат: система була впроваджена вчасно, але в кілька хвиль (release waves), щоб мінімізувати ризики.

Приклад 2. Державна система електронного урядування (рис.2.42)

Контекст: багаторічна програма цифровізації публічних послуг.

PMBOK: ключовим стало управління стейкхолдерами (міністерства, громадяни, постачальники ІТ-послуг).

Як застосували PMBOK:

✓ Управління зацікавленими сторонами – різні міністерства та відомства мали конфліктні інтереси; застосували матрицю Power-Interest для управління.

✓ Управління якістю – створили незалежну QA-групу для перевірки функціоналу перед релізом.

✓ Управління постачаннями – більшість робіт виконували через тендери; суворо контролювали контракти.

Результат: проєкт реалізовувався поступово, але стандарти PMBOK дозволили уникнути хаосу в управлінні сотнями підрядників, завдяки матриці комунікацій вдалося уникнути хаосу та узгодити інтереси різних сторін

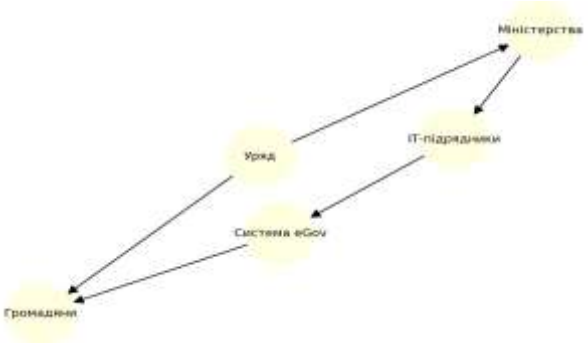

Рисунок 2.42 – Схема взаємодії стейкхолдерів у держсистемі.

Приклад 3. Розробка великої телеком-системи (рис.2.43)

Контекст: оператор мобільного зв'язку запускав нове покоління мережі (5G).

PMBOK: акцент на управлінні ризиками та контролі якості, бо навіть невелика помилка могла коштувати мільйони.

Як застосували PMBOK:

✓ Управління часом – застосували метод критичного шляху (CPM) для узгодження будівництва веж, налаштування обладнання й запуску софту.

✓ Управління ресурсами – паралельне залучення сотень інженерів у різних регіонах.

✓ Моніторинг і контроль – щотижневі дашборди з ключовими показниками (KPI).

Результат: завдяки структурованому підходу розгортання мережі відбулося синхронно з рекламною кампанією.

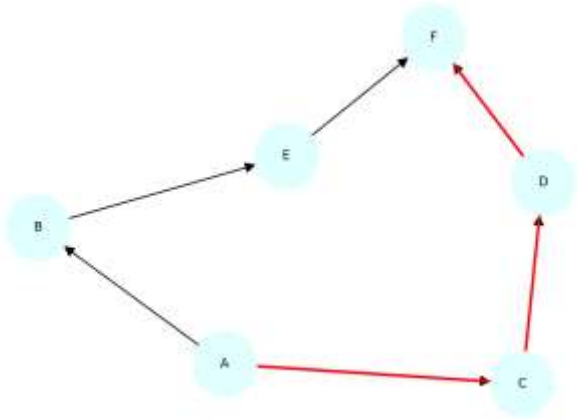

Рисунок 2.43 – Критичний шлях (CPM) для телеком-проєкту.

Висновок.

У великих IT-програмах PMBOK виступає рамковою методологією, яка забезпечує:

✓ структурованість;

✓ контроль над ризиками, змінами й ресурсами;

- ✓ зменшення конфліктів між стейкхолдерами;
- ✓ документованість і прозорість процесів.

Саме тому PMBOK часто використовується в державних, телекомунікаційних та корпоративних програмах, де помилка може коштувати мільйони.

2.3.5. Порівняння PMBOK з іншими стандартами

PMBOK – не єдиний стандарт управління проєктами. У світі активно застосовуються:

- ✓ ISO 21500 – міжнародний стандарт, орієнтований на узгодження термінології та принципів.
- ✓ PRINCE2 – британський стандарт, який наголошує на процесах та ролях.

PMBOK

- ✓ Автор: Project Management Institute (PMI, США).
- ✓ Орієнтація: знання (Knowledge Areas) та процесні групи.
- ✓ Підхід: рамковий, дає набір інструментів і практик.
- ✓ Сфера: від IT і будівництва до оборонних і держпроєктів.

ISO 21500

- ✓ Автор: International Organization for Standardization.
- ✓ Орієнтація: уніфікація термінології та узагальнення практик.
- ✓ Підхід: більш загальний, ніж PMBOK, не деталізує інструменти.
- ✓ Сфера: міжнародні організації, державні структури.

PRINCE2

- ✓ Автор: UK Office of Government Commerce (OGC).
- ✓ Орієнтація: процесна модель управління проєктами.
- ✓ Підхід: чітко визначені ролі, етапи, контрольні точки.
- ✓ Сфера: державні проєкти у Великій Британії та ЄС, великий бізнес.

На рисунку 2.44 зображена діаграма Венна взаємодії стандартів та в таблиці 2.5 – їх порівняння.

Рисунок 2.44 – Діаграма Венна (PMBOK – ISO – PRINCE2: спільне та відмінне).

Таблиця 2.5 – Основні відмінності

Критерій	PMBOK	ISO 21500	PRINCE2
Орієнтація	Знання + процеси	Принципи та рекомендації	Процеси та ролі
Глибина	Деталізований, 10 областей знань	Загальний, високий рівень	Дуже структурований, покроковий
Гнучкість	Висока	Середня	Низька (чітко прописана методика)
Популярність	США, світ	Міжнародні організації	ЄС, Великобританія
Застосування у ІТ	Дуже поширене	Частіше для узгодження стандартів	У великих організаціях ЄС

Висновок:

✓ PMBOK – дає набір інструментів, які можна адаптувати.

✓ ISO 21500 – створює міжнародну «мову управління проєктами».

✓ PRINCE2 – дисциплінує процес, але менш гнучкий.

У практиці ІТ-компаній часто комбінують PMBOK із Agile підходами, а PRINCE2 більше застосовують у великих державних програмах.

2.4. Сильні сторони традиційних методів управління проєктами

1. Висока формалізація

Традиційні методи (Waterfall, PMBOK) передбачають чітку послідовність кроків та суворе документування кожного етапу (рис.2.45).

✓ Усі вимоги, специфікації та контракти фіксуються письмово.

✓ Прийняття рішень відбувається через офіційні протоколи й затвердження.

✓ Це зменшує ризик непорозумінь між замовником і виконавцем.

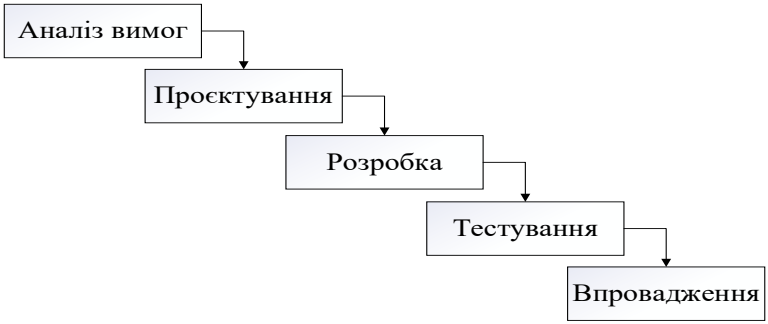

Рисунок 2.45 – Етапи традиційних методів

Приклад: у державних ІТ-системах (наприклад, електронне урядування) наявність офіційної документації гарантує прозорість і можливість перевірки аудиторами.

2. Контроль і прогнозованість

✓ Завдяки каскадній структурі кожна фаза має чіткі вхідні й вихідні дані.

✓ Менеджер може прогнозувати терміни, бюджет і ресурси ще на початку.

✓ Використання Gantt-діаграм та методів критичного шляху (CPM) дозволяє побудувати точний графік (рис.2.46).

Приклад: у розробці банківських систем, де будь-яке відхилення може призвести до серйозних втрат, традиційний підхід дозволяє уникати сюрпризів.

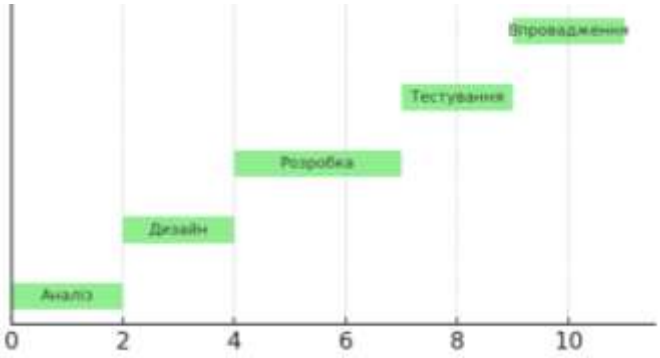

Рисунок 2.46 – Gantt-діаграма для ілюстрації прогнозованості проєкту

3. Юридична та контрактна відповідність

Традиційні методи добре узгоджуються з юридичними процедурами: тендерами, державними контрактами, аудитами (рис.2.47).

✓ У таких проєктах обов'язково ведеться велика кількість документів (технічне завдання, план управління, звіти).

✓ Це робить метод підходящим для проектів, де існує жорсткий контроль з боку регуляторів.

Приклад: у військових або космічних проєктах формалізація потрібна для гарантії якості та відповідності стандартам безпеки.

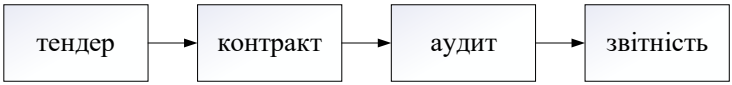

Рисунок 2.47 – Ілюстрація контрактного циклу.

2.5. Слабкі сторони традиційних методів

1. Низька гнучкість

Традиційні методи (Waterfall, PMBOK у строгому застосуванні) мають чітку послідовність етапів, що ускладнює внесення змін:

✓ Якщо замовник змінює вимоги, потрібно переглядати вже затверджені документи.

✓ Вартість таких змін зростає експоненційно на пізніх стадіях (рис.2.48).

✓ Це робить модель менш придатною для динамічного середовища ІТ.

Приклад: у розробці мобільного застосунку, якщо з'являється нова версія ОС, адаптація у Waterfall-проєкті може вимагати повного перегляду плану.

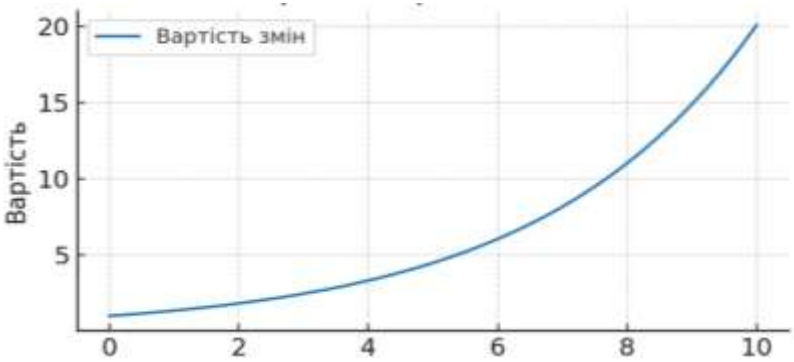

Рисунок 2.48 – Крива вартості змін (чим пізніше внесено зміну – тим дорожче).

2. Повільне реагування на зміни (рис.2.49):

✓ Між початковим аналізом і фінальною реалізацією може пройти рік чи більше.

✓ За цей час ринок або технології суттєво змінюються.

✓ Традиційні методи не передбачають швидкого фідбеку від користувачів.

Приклад: у банківських системах вимоги безпеки можуть змінюватися швидше, ніж рухається проєкт, що призводить до затримок.

Рисунок 2.49 – Схема часової затримки в традиційних методах

3. Високий ризик невідповідності кінцевого продукту потребам користувача (рис.2.50):

✓ Замовник часто бачить готову систему лише у фіналі.

✓ Якщо на етапі аналізу були допущені помилки, виправити їх після реалізації складно і дорого.

✓ Це створює ризик, що кінцевий продукт не буде відповідати очікуванням користувача.

Приклад: державний портал, який після запуску виявився непрактичним для громадян через відсутність UX-досліджень на початку.

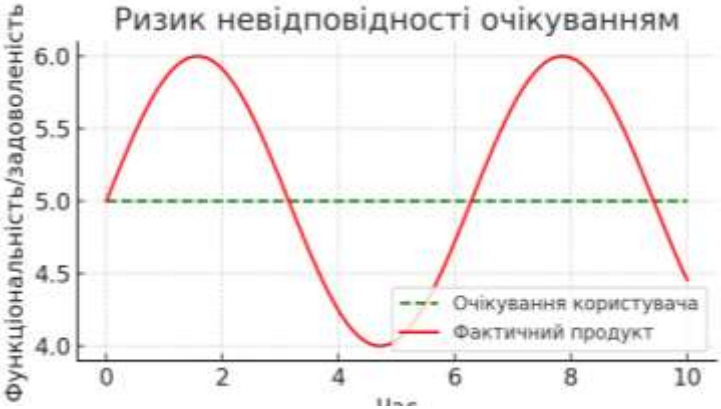

Рисунок 2.50 – Діаграма ризику невідповідності
(очікування користувача vs фінальний продукт).

2.6. Порівняння з Agile

Традиційні підходи (Waterfall, PMBOK у класичній формі) і Agile-методології (Scrum, Kanban, XP) мають різну філософію управління

проєктами (табл.2.6, рис.2.51-2.53).

✓ Традиційні методи орієнтовані на документацію, формалізацію та прогнозованість.

✓ Agile орієнтований на гнучкість, адаптацію та швидку взаємодію з користувачем.

Таблиця 2.6 – Ключові відмінності

Критерій	Традиційні методи (Waterfall, PMBOK)	Agile-підходи (Scrum, Kanban)
Планування	Повне на початку	Ітеративне, у спринтах
Фокус	Документація	Робочий продукт
Зміни	Важкі, дорогі	Приймаються у будь-який момент
Зворотний зв'язок	Наприкінці проєкту	Постійний, після кожної ітерації
Ролі	Формальні (PM, аналітики, інженери)	Гнучкі (Scrum Master, команда)
Контроль	Через звіти, документи	Через демо, ретроспективи
Ризик невідповідності	Високий	Знижений завдяки фідбеку
Застосування	Банки, держпроєкти, оборонка	ІТ-продукти, стартапи, UX-heavy системи

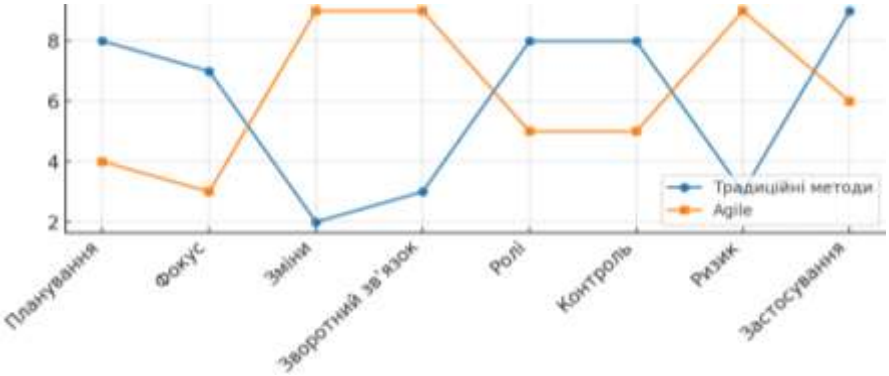

Рисунок 2.51 – Діаграма порівняння (традиційні vs Agile).

Рисунок 2.52 – Вісь «Формалізація ↔ Гнучкість» із розташуванням Waterfall, PMBOK, Scrum, Kanban.

PMBOK (структури, процеси)	Hybrid Model	Agile (гнучкість, ітерації)

Рисунок 2.53 – Інфографіка Hybrid Model (Agile + PMBOK).

Приклади застосування:

✓ Waterfall / PMBOK: будівництво дата-центрів, ERP-системи, державні портали.

✓ Agile: розробка мобільних додатків, SaaS-рішень, стартапів з невизначеними вимогами.

Висновки

✓ Традиційні методи корисні там, де потрібна жорстка регуляція та контроль.

✓ Agile краще підходить для динамічного середовища та користувацько-орієнтованих продуктів.

✓ У сучасній практиці часто використовують гібридні моделі (Agile+PMBOK).

2.7. Практичні кейси: коли традиційний підхід кращий за Agile

Загальний підхід вибору між Agile та Традиційні наведений на рис.2.54. Розглянемо використання у певних напрямках:

1. Державні проєкти:

✓ Урядові IT-системи (електронні реєстри, податкові бази, e-Gov).

✓ Потребують жорсткої регламентації, прозорості, аудиту.

✓ Agile не може забезпечити належний рівень формалізації для тендерів і контрактів.

Матриця вибору: Agile vs Традиційні

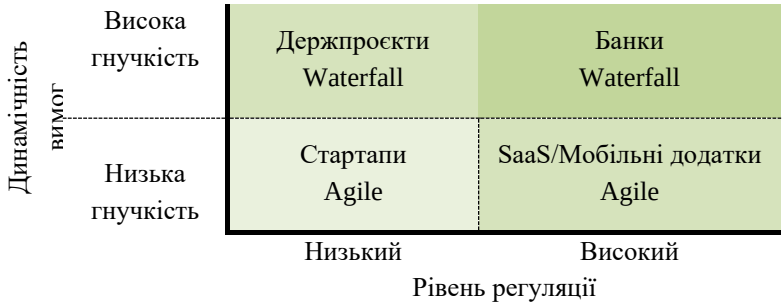

Рисунок 2.55 – Матриця вибору: Agile vs Традиційні (по осях "динамічність вимог" – "рівень регуляції").

Приклад: створення Єдиного державного демографічного реєстру в Україні (водійські права, паспорти).

2. Військові та оборонні розробки:

✓ Тут пріоритетом є безпека, відповідність стандартам та контроль.

✓ Кожна зміна повинна бути задокументована і схвалена.

✓ Agile з його швидкими ітераціями не завжди відповідає високим вимогам безпеки.

Приклад: програмне забезпечення для систем управління озброєнням.

3. Банківські та фінансові системи:

✓ Банки працюють у регульованому середовищі, де потрібна повна звітність та контроль.

✓ Традиційні методи дозволяють детально планувати проєкт і забезпечити відповідність закону.

Приклад: впровадження системи core-banking або інтеграція SWIFT.

4. Інфраструктурні та апаратні проєкти:

✓ Якщо продукт складається не лише з ПЗ, а й апаратної частини (сервери, дата-центри, мережі).

✓ Тут зміни коштують надзвичайно дорого, тому важлива точність планування.

Приклад: будівництво дата-центру з інтегрованою ERP-системою.

5. Довгострокові мегапроєкти:

✓ Проєкти тривалістю 3-5 років і більше.

✓ Agile важко масштабувати на сотні учасників без втрати керованості.

✓ Традиційні методи забезпечують стійкість і контроль у великій програмі.

Приклад: міжнародні проєкти цифровізації залізничних перевезень.

Краще Waterfall/PMBOK застосовувати для таких напрямків:

✓ Мегапроєкти

✓ Інфраструктура

✓ Банківські системи

✓ Військові розробки

✓ Держпроєкти

Висновок. Традиційний підхід не є застарілим – він просто використовується у тих сферах, де Agile втрачає ефективність. Це проєкти з високою ціною помилки, великим масштабом та жорсткими регуляторними вимогами.

2.8. Загальні висновки

2.8.1. Основні тези

1. Ключові ідеї про Waterfall і PMBOK:

✓ Waterfall – класична каскадна модель, що забезпечує чітку послідовність етапів: від аналізу вимог до впровадження.

✓ PMBOK – міжнародний стандарт, який описує процесні групи та області знань управління проєктами, формує основу професійної діяльності менеджера.

✓ Обидва підходи орієнтовані на структурованість, документацію та контроль.

2. Місце методів в сучасному управлінні IT-проєктами.

Попри поширення Agile, традиційні методи не втратили актуальності. Вони залишаються незамінними у сферах, де потрібні:

✓ жорстка регуляція,

✓ повна звітність,

- ✓ високий рівень безпеки,
- ✓ прогнозованість у великих програмах.

✓ Waterfall і PMBOK застосовуються у державних, військових, фінансових та інфраструктурних проєктах.

3. Як класичні та гнучкі методи можуть доповнювати одне одного

Сучасна практика часто поєднує Agile та традиційні стандарти у гібридних моделях (рис.2.55):

✓ PMBOK задає рамку управління: інтеграція, ризики, ресурси, стейкхолдери.

✓ Agile забезпечує гнучкість у розробці та швидкий фідбек від користувачів.

Рисунок 2.55 – Де краще класичні, де – гнучкі, де – комбінація.

Такий підхід дозволяє одночасно мати прозорість і контроль та адаптивність і швидкість.

Висновок. Waterfall і PMBOK не є конкурентами Agile, а швидше – його доповненням. Разом вони створюють ефективний набір інструментів для управління сучасними ІТ-проєктами.

2.8.2. Питання для самоперевірки

1. У чому полягає сутність каскадної моделі (Waterfall)?
2. Хто і коли вперше запропонував ідею Waterfall?
3. Які основні фази містить каскадна модель?
4. У чому переваги Waterfall для великих державних і військових проєктів?
5. Що таке PMBOK і яка роль PMI у його розвитку?
6. Назвіть п’ять процесних груп PMBOK.
7. Які десять областей знань виділяє PMBOK?
8. Чим відрізняється управління ризиками у PMBOK від підходів Agile?
9. Які сильні сторони традиційних методів управління проєктами?
10. Які слабкі сторони Waterfall і PMBOK?
11. У яких випадках Agile дає кращий результат, ніж традиційні підходи?
12. Які галузі переважно використовують традиційні методи управління?

2.8.3. Завдання для самостійної роботи

1. Практичне завдання 1. Побудуйте схему Waterfall-моделі для проєкту створення мобільного застосунку (наприклад, мобільний банкінг). Визначте ключові етапи, учасників та очікувані результати кожної фази.

2. Практичне завдання 2. Виберіть конкретний кейс (наприклад, розробка електронного журналу для університету).

✓ Опишіть, як би ви реалізували його, використовуючи підходи PMBOK.

✓ Опишіть, як би виглядав той самий проєкт за допомогою Agile (Scrum/Kanban).

✓ Порівняйте отримані результати та визначте, які сильні й слабкі сторони проявляться у кожному випадку.

3. Практичне завдання 3 (поглиблене). Побудуйте таблицю порівняння для конкретного кейсу (наприклад, розробка державного порталу), у якій порівняйте Waterfall, PMBOK і Agile за критеріями:

- ✓ час реакції на зміни,
- ✓ рівень контролю,
- ✓ гнучкість,
- ✓ прозорість для замовника,
- ✓ вартість внесення змін.

Шаблон схеми Waterfall (порожня діаграма для заповнення студентами).

Таблиця для порівняння PMBOK та Agile.

Критерій	Waterfall	PMBOK	Agile
Час реакції			
Рівень контролю			
Гнучкість			
Прозорість			
Ваптість змін			

Вісь "Формалізація – Гнучкість" для самостійного розташування методів.

Ці завдання допоможуть закріпити матеріал і навчитися застосовувати теоретичні знання на практиці.

2.9. Контрольні запитання

1. Що таке модель Waterfall? Хто і в який час її запропонував?
2. Назвіть і охарактеризуйте основні фази каскадної моделі управління проєктом.
3. У яких галузях і для яких проєктів традиційна модель Waterfall залишається затребуваною? Обґрунтуйте.
4. Чому, незважаючи на розвиток Agile, стандарт PMBOK досі широко застосовується в IT-галузі?
5. Що таке PMBOK і яку роль відіграє Project Management Institute (PMI) в його розробці?
6. Назвіть п'ять процесних груп PMBOK та коротко опишіть кожну з них.
7. Охарактеризуйте фазу ініціації проєкту за PMBOK: цілі, ключові документи, результати.
8. Що включає фаза планування за PMBOK? Назвіть основні процеси та артефакти.
9. У чому полягає суть фази виконання проєкту за PMBOK та які ключові дії здійснюються в її межах?
10. Що таке моніторинг і контроль у PMBOK? Які метрики та інструменти використовуються?
11. Охарактеризуйте фазу завершення проєкту за PMBOK: дії, документи, уроки засвоєні.
12. Назвіть десять областей знань PMBOK та стисло поясніть, що охоплює кожна з них.
13. У чому полягає управління ризиками як область знань PMBOK? Опишіть основні процеси.
14. Як PMBOK підходить до управління зацікавленими сторонами (Stakeholder Management)?
15. Порівняйте PMBOK з іншими стандартами управління проєктами (наприклад, PRINCE2, ISO 21500): спільне та відмінне.

3. ГНУЧКІ МЕТОДОЛОГІЇ УПРАВЛІННЯ ІТ-ПРОЄКТАМИ (AGILE, SCRUM, KANBAN)

3.1. Вступ

Сучасна ІТ-галузь є однією з найбільш динамічних сфер людської діяльності. Розвиток цифрових технологій, глобальна конкуренція та швидке зростання потреб користувачів призводять до того, що вимоги до програмних продуктів змінюються майже щодня. Якщо в 1980-1990-х роках життєвий цикл програмного забезпечення міг обчислюватися десятиліттями, то сьогодні середній час «актуальності» будь-якого продукту скоротився до кількох років, а інколи навіть місяців.

3.1.1. Актуальність гнучких методів в ІТ

У цих динамічних умовах використання традиційних каскадних методів управління проєктами виявилося недостатньо ефективним. Модель Waterfall, яка передбачає послідовне проходження фаз (аналіз → проєктування → розробка → тестування → впровадження), добре підходить для стабільних і чітко визначених систем, але не витримує темпів сучасного ринку. Основні проблеми, з якими стикнулися ІТ-компанії при застосуванні класичних підходів:

- ✓ Велика тривалість проєктів. Програмне забезпечення часто ставало застарілим ще до моменту його завершення.
- ✓ Обмежений вплив клієнта. Замовник брав участь лише на етапі формування вимог, тому кінцевий продукт міг не відповідати актуальним бізнес-потребам.
- ✓ Низька гнучкість. Зміна навіть однієї вимоги призводила до затримок і додаткових витрат, оскільки доводилося перепроєктовувати значні частини системи.

Розвиток інтернету, поява мобільних технологій і хмарних сервісів ще більше підсилили проблему. Компанії, які не встигали швидко адаптуватися до змін, втрачали конкурентоспроможність. Саме це й стало каталізатором пошуку нових підходів до управління ІТ-проєктами.

Agile-методології з’явилися як відповідь на потребу:

✓ Скоротити час виходу продукту на ринок. Замість очікування роками користувач отримує перший робочий результат уже через кілька тижнів.

✓ Забезпечити гнучкість. Зміни у вимогах інтегруються прямо в процес розробки.

✓ Посилити роль клієнта. Замовник стає активним учасником розробки, а не пасивним спостерігачем.

✓ Зменшити ризики. Завдяки частим ітераціям і швидкому фідбеку виявлення проблем відбувається на ранніх етапах.

Сьогодні Agile є стандартом де-факто в індустрії програмного забезпечення. Більшість IT-компаній світу – від стартапів до гігантів на кшталт Google, Microsoft чи Spotify – використовують ті чи інші гнучкі методології (Scrum, Kanban, XP тощо).

Таким чином, актуальність Agile зумовлена самим характером IT-галузі: швидкі зміни, зростаюча складність продуктів і висока конкуренція роблять гнучкі методи не просто корисними, а необхідними для виживання на ринку.

3.1.2. Проблеми традиційних моделей, які вирішує Agile

Традиційні підходи до управління проєктами, зокрема Waterfall та інші каскадні моделі, були ефективними у середині XX століття, коли більшість інженерних проєктів мали відносно стабільні вимоги та передбачуване середовище. Проте з часом, особливо у сфері розробки програмного забезпечення, почали проявлятися суттєві обмеження.

1. Довгий цикл розробки

У Waterfall користувач бачить готовий продукт лише після проходження всіх фаз: від аналізу вимог до фінального впровадження. Це може займати роки.

Проблема: за цей час ринок і бізнес-потреби кардинально змінюються.

Рішення Agile: розробка ведеться ітераціями (спринтами), кожна з яких приносить частину робочого функціоналу. Замовник отримує результат уже через 2-4 тижні.

2. Відрив від замовника

У класичних моделях замовник залучений лише на початку (формування вимог) і в кінці (приймання продукту).

Проблема: кінцевий результат може не відповідати реальним потребам бізнесу.

Рішення Agile: замовник – активний учасник процесу, регулярно переглядає результати і коригує бачення продукту.

3. Високі ризики невідповідності продукту

Чим довший життєвий цикл розробки, тим більше шансів, що продукт застаріє ще до випуску або не відповідатиме потребам ринку.

Проблема: великі інвестиції можуть бути витрачені даремно.

Рішення Agile: завдяки частим релізам і постійному фідбеку ризики знижуються, а команда швидко виправляє курс.

4. Повільне реагування на зміни

У Waterfall внесення змін вимагає перегляду документації та планів усіх наступних фаз, що є дорогим і затяжним процесом.

Проблема: зміни бізнес-середовища «ламають» увесь план.

Рішення Agile: гнучка інтеграція змін у поточну ітерацію чи наступний спринт без зупинки всього проєкту.

5. Надмірна бюрократія і документація

Класичні методи передбачають створення великого обсягу технічних завдань, планів та звітів.

Проблема: багато документів залишаються невикористаними, сповільнюючи роботу.

Рішення Agile: використовується лише мінімально необхідна документація, що безпосередньо допомагає створювати продукт.

Таким чином, Agile виник як альтернатива традиційним методам, щоб вирішити найголовніші проблеми:

- ✓ занадто довгі цикли розробки,
- ✓ відсутність гнучкості,
- ✓ недостатня залученість клієнта,
- ✓ високі ризики невідповідності кінцевого продукту потребам.

Завдяки Agile програмні продукти створюються швидше, ближче до вимог замовника та з меншими ризиками для бізнесу.

3.2. Agile як філософія управління

Agile – ідеологія гнучкого управління проєктами.

3.2.1. Історія виникнення Agile

У 1980-1990-х роках IT-індустрія переживала справжню «кризу управління проєктами». За даними Standish Group (Chaos Report), понад 50 % програмних проєктів завершувалися із серйозними проблемами: перевищенням бюджету, зривами строків або невідповідністю очікуванням клієнтів. Близько 30 % проєктів взагалі провалювалися і ніколи не впроваджувалися.

Основними причинами були:

- ✓ Жорсткі каскадні методи (Waterfall). Вони вимагали чітко визначених вимог на початку, що у швидкозмінному світі було майже неможливо.
- ✓ Тривалі цикли розробки. Програмне забезпечення створювалося роками, а коли доходило до користувача – воно вже застарівало.
- ✓ Відсутність залучення замовника. Замовник фактично не впливав на продукт після старту проєкту.
- ✓ Складність великих систем. Чим більше було проєктів, тим вище була ймовірність провалу.

Ця ситуація вимагала принципово нового підходу.

В лютому 2001 року 17 відомих експертів у сфері програмної інженерії зібралися в гірському курорті Сноуберд (штат Юта, США). Серед них були:

- ✓ Кент Бек (автор Extreme Programming),
- ✓ Джефф Сазерленд і Кен Швабер (співавтори Scrum),
- ✓ Алістер Коберн (Crystal Clear),
- ✓ Мартін Фаулер (експерт з об’єктно-орієнтованого програмування),
- ✓ та інші.

Метою зустрічі було обговорення проблем традиційного управління і пошук спільної філософії, яка б поєднала позитивні ідеї існуючих «легковагових методологій».

Результатом стало створення документа Agile Manifesto (зустріч у Сноуберді 2001 р.) – Маніфест гнучкої розробки ПЗ (рис.3.1).

У Маніфесті було сформульовано:

✓ 4 цінності (ми їх уже розглядали: люди > процеси, продукт > документація, клієнт > контракт, зміни > план).

✓ 12 принципів (гнучкість, інкрементальність, самоорганізація, простота тощо).

Рисунок 3.1 – Виникнення Agile Manifesto (зустріч у Сноуберді в 2001 р.)

Цей документ став основою нової філософії, яка згодом отримала назву Agile (гнучка розробка).

Після оприлюднення Маніфесту Agile швидко поширився в ІТ-компаніях США, Європи та Японії. З'явилися нові практики і методології, що базуються на Agile:

✓ Scrum – найпопулярніша методика управління командами.

✓ Kanban – підхід, натхненний японською виробничою системою Toyota.

✓ Extreme Programming (XP) – фокус на якості коду і технічній досконалості.

✓ Lean Software Development – адаптація принципів Lean до ІТ.

Agile став не лише набором практик, а цілою культурою управління проєктами. У 2010-х роках Agile почав застосовуватися не лише в ІТ, але й у маркетингу, освіті, управлінні бізнесом і навіть у державному секторі.

Висновки

✓ Agile виник як відповідь на кризу традиційних моделей управління

проєктами.

✓ Його створення було колективним результатом об'єднання різних «легких методологій».

✓ Agile Manifesto (2001) заклав фундамент для сучасного підходу до управління.

✓ Agile сьогодні є стандартом де-факто для IT-індустрії та активно поширюється в інші сфери.

3.2.2. Основні принципи Agile Manifesto

У 2001 році під час зустрічі 17 провідних експертів у галузі розробки програмного забезпечення було прийнято документ, який отримав назву Agile Manifesto. Він став відправною точкою для нової епохи в управлінні IT-проєктами.

Маніфест визначив не лише 4 ключові цінності, але й 12 принципів, які конкретизують, як саме ці цінності мають реалізовуватись у практиці.

У центрі Agile лежать чотири базові цінності, сформульовані в Agile Manifesto (2001). Вони визначають, чим у роботі над проєктами варто керуватися в першу чергу.

1. Люди та взаємодія важливіші за процеси та інструменти.

✓ Програмне забезпечення створюють люди, а не процеси чи інструменти.

✓ Найкращий результат досягається тоді, коли команда ефективно комунікує та співпрацює.

2. Працюючий продукт важливіший за вичерпну документацію.

✓ Документи мають значення, але головне – це реальний результат, який можна протестувати та використовувати.

✓ Продукт має з'являтися якомога раніше у вигляді робочих версій.

3. Співпраця з клієнтом важливіша за контрактні зобов'язання.

✓ Контракт важливий, але ключове – живий діалог із замовником і швидка реакція на його змінені потреби.

✓ Успіх визначається задоволенням користувача, а не тим, наскільки ми «виконали контракт».

4. Реагування на зміни важливіше за дотримання плану.

✓ У динамічному середовищі ІТ жорсткий план стає обмеженням.

✓ Agile закликає бачити в змінах не загрозу, а можливість створити кращий продукт.

Ці тези окреслюють загальний вектор мислення, але щоб перетворити їх на конкретні практики, було сформульовано 12 принципів Agile.

Цінності конкретизуються через 12 принципів (рис.3.2), що визначають щоденну практику команд:

1. Найвищий пріоритет – задовольнити замовника завдяки ранній і безперервній поставці ПЗ.
2. Вітати зміни вимог, навіть на пізніх етапах.
3. Часто постачати працююче ПЗ (від кількох тижнів до кількох місяців).
4. Бізнес-замовники і розробники мають співпрацювати щодня.
5. Будувати проєкти навколо мотивованих людей.
6. Найефективніше передавати інформацію особистим спілкуванням.
7. Працюючий продукт – головна міра прогресу.
8. Agile-процеси підтримують сталий розвиток і постійний темп.
9. Постійна увага до технічної досконалості й дизайну.
10. Простота – мистецтво робити лише потрібне.
11. Самоорганізовані команди створюють найкращі рішення.
12. Команда регулярно аналізує свою роботу й вдосконалюється.

Ці 12 принципів Agile мають сучасну інтерпретацію:

1. Найвищий пріоритет – задовольнити замовника через ранню та безперервну поставку цінного програмного забезпечення.
2. Вітати зміни вимог, навіть на пізніх етапах розробки. Agile-процеси обертають зміни на користь клієнта.
3. Часто постачати працююче ПЗ, від кількох тижнів до кількох місяців, з перевагою коротших циклів.
4. Бізнес-замовники та розробники повинні працювати разом щодня протягом усього проєкту.
5. Будувати проєкти навколо мотивованих людей. Створювати умови, підтримку та довіру, щоб команда могла виконати роботу.
6. Найефективніший спосіб передачі інформації у команді –

безпосереднє спілкування «обличчям до обличчя».

7. Працюючий продукт – головна міра прогресу.

8. Agile-процеси сприяють сталому розвитку. Команда, замовники та користувачі повинні підтримувати постійний темп без перевантажень.

9. Безперервна увага до технічної досконалості та якісного дизайну підвищує гнучкість.

10. Простота – мистецтво мінімізувати роботу, яку не потрібно виконувати.

11. Найкращі архітектури, вимоги та проєкти створюються самоорганізованими командами.

12. Команда регулярно аналізує свою роботу, шукає способи підвищення ефективності та відповідно коригує свою поведінку.

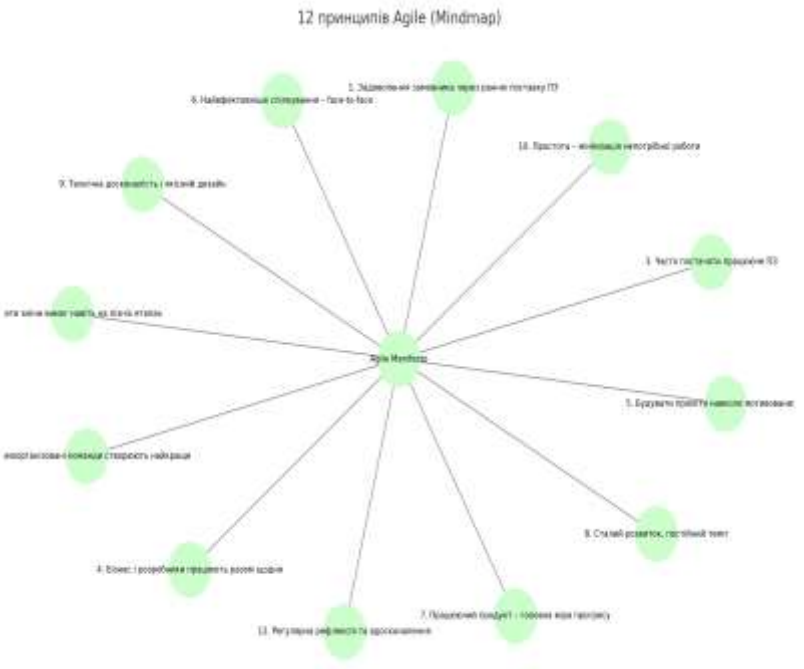

Рисунок 3.2 – Mindmap «12 принципів Agile»

Значення принципів Agile для ІТ.

Ці принципи не є жорсткими інструкціями, вони радше відображають філософію роботи в умовах змін. Вони допомагають:

- ✓ підтримувати баланс між плануванням і гнучкістю,
- ✓ забезпечувати орієнтацію на клієнта, а не на процес,
- ✓ стимулювати самоорганізацію та відповідальність команд,
- ✓ створювати продукти, що швидко відповідають потребам ринку.

Взаємодію 4 цінностей з 12 принципами зображено на рис.3.3.

Рисунок 3.3 – Схема взаємодії «4 цінності → 12 принципів».

Ця схема демонструє, як із чотирьох ключових цінностей Agile Manifesto випливають дванадцять принципів, що конкретизують практичну реалізацію гнучких методів.

У результаті Agile перестає бути лише методологією – це стає культурою управління проєктами, яка змінила світ ІТ.

Висновки:

✓ 4 цінності задають напрямок мислення,

✓ 12 принципів дають орієнтири для щоденної роботи,

✓ Разом вони формують філософію Agile, яка дозволяє створювати продукти, орієнтовані на потреби користувачів, у динамічному середовищі.

3.2.3. Ключові відмінності Agile від Waterfall

1. Структура процесу

Waterfall: робота організована у вигляді послідовних фаз (аналіз → проєктування → розробка → тестування → впровадження). Перехід до наступної фази можливий лише після завершення попередньої.

Agile: процес поділений на ітерації (спринти) тривалістю 1-4 тижні. У кожній ітерації команда проходить усі етапи – від аналізу до тестування – і створює частину робочого продукту.

2. Гнучкість до змін

Waterfall: будь-які зміни вимог після затвердження документації – це критичний ризик, що веде до додаткових витрат і затримок.

Agile: зміни сприймаються як природна частина процесу. Нові вимоги інтегруються у наступні спринти.

3. Участь замовника

Waterfall: замовник бере участь лише на початку (постановка вимог) і в кінці (приймання продукту).

Agile: замовник активно залучений протягом усього проєкту – бере участь у плануванні, оглядах спринтів, дає зворотний зв'язок.

4. Ризики та прогнозованість

Waterfall: високий рівень ризику – кінцевий продукт замовник бачить лише через кілька місяців чи років.

Agile: ризики знижуються за рахунок ранньої демонстрації результатів та швидкого фідбеку.

5. Документація

Waterfall: обов’язкова велика кількість специфікацій, планів, технічних завдань.

Agile: документація зведена до мінімуму – лише та, що дійсно потрібна

команді та замовнику.

6. Орієнтація на результат

Waterfall: успіх вимірюється тим, наскільки точно команда дотрималася початкового плану.

Agile: успіх визначається тим, наскільки продукт задовольняє актуальні потреби користувача.

7. Командна динаміка

Waterfall: команди зазвичай мають жорстку ієрархію: менеджери → аналітики → розробники → тестувальники.

Agile: команди самоорганізовані, ролі гнучкі (наприклад, у Scrum є Product Owner, Scrum Master та кросфункціональна команда).

Загальне порівняння наведена у таблиці 3.1.

Таблиця 3.1 – Порівняльна таблиця Agile та Waterfall

Критерій	Waterfall	Agile
Структура процесу	Послідовні фази	Ітеративні спринти
Реакція на зміни	Складна і дорога	Легка інтеграція
Участь замовника	На початку і в кінці	Постійна участь
Демонстрація продукту	Наприкінці проєкту	Після кожного спринту
Документація	Обов'язкова, велика кількість	Мінімально необхідна
Вимірювання успіху	Відповідність плану	Задоволеність користувача
Динаміка команди	Ієрархічна	Самоорганізована

Висновки:

✓ Agile і Waterfall не є взаємовиключними, але мають різну логіку.

✓ Waterfall підходить для стабільних і передбачуваних проєктів.

✓ Agile підходить для динамічних умов і швидкозмінних вимог.

✓ У сучасних ІТ-компаніях все частіше застосовуються гібридні моделі, де планування поєднується з ітеративною розробкою.

3.3. Scrum

Scrum – універсальний підхід для управління саме ІТ-проєктами.

3.3.1. Історія виникнення Scrum

Scrum з'явився у 1990-х роках як відповідь на виклики, з якими стикалися ІТ-компанії у розробці складних програмних продуктів.

Термін "Scrum" уперше використали Хіротака Такеучі та Ікуджіро Нонака у 1986 році у статті "The New New Product Development Game" (журнал Harvard Business Review). Вони описали новий підхід до створення продуктів, у якому команда працює як єдиний механізм, схожий на «скрам» у регбі – коли гравці об'єднуються у щільну групу для просування вперед.

У 1995 році Кен Швабер і Джефф Сазерленд представили першу офіційну версію Scrum на конференції OOPSLA (Object-Oriented Programming Systems, Languages & Applications). Вони систематизували метод, виділили ключові ролі, артефакти й події, що стали основою сучасного Scrum Guide.

Сьогодні Scrum – один із найпоширеніших фреймворків Agile у світі, яким користуються компанії від стартапів до міжнародних корпорацій.

Scrum – це гнучкий фреймворк управління проєктами, який дозволяє створювати продукти ітеративно та інкрементально (рис.3.4). Він не є жорсткою методологією з правилами «раз і назавжди», а радше набіром практик і принципів, які команда може адаптувати під свої потреби.

Основні характеристики Scrum:

✓ Ітеративність. Робота ведеться короткими циклами – спринтами (1-4 тижні).

✓ Інкрементальність. Кожен спринт завершується створенням частини готового продукту.

✓ Прозорість. Усі учасники бачать прогрес завдяки беклогам, щоденним зустрічам, оглядам.

✓ Співпраця. У Scrum немає класичного «керівника». Команда самоорганізується, а ролі Product Owner і Scrum Master лише допомагають.

✓ Адаптивність. Scrum дозволяє швидко реагувати на зміни вимог, що особливо важливо у сучасному ІТ.

Рисунок 3.4 – Зображення воронки пріоритезації беклогу продукту

3.3.2. Scrum у практиці

Scrum використовується для:

✓ розробки програмного забезпечення (особливо складних систем із багатьма невідомими);

✓ стартапів, де важливо швидко тестувати ідеї на ринку;

✓ великих організацій, які прагнуть підвищити швидкість і якість продуктів;

✓ не тільки в IT, але й у маркетингу, освіті, R&D, навіть у сфері державного управління.

3.3.3. Scrum: ролі в методології

Scrum – це одна з найпопулярніших і найпоширеніших методологій у

світі Agile. Його особливість полягає в тому, що він дуже чітко визначає ролі, артефакти та події. У центрі Scrum стоїть команда, яка самоорганізовується й працює ітераціями – «спринтами».

У Scrum існує три ключові ролі, розглянемо кожну з ролей.

1. Product Owner (Власник продукту).

Відповідає за цінність продукту, який створює команда. Управляє Product Backlog (списком усіх вимог, функцій і задач для продукту). Пріоритизує завдання: визначає, які функції потрібно реалізувати першими, а які можна відкласти. Є «голосом замовника» в команді – представляє потреби бізнесу та кінцевих користувачів. Приймає роботу команди після завершення спринтів, підтверджує, що результат відповідає очікуванням.

Приклад: у проєкті створення мобільного банківського застосунку Product Owner може вирішити, що функція «онлайн-перекази» важливіша за «інвестиційні сервіси», і саме вона має бути реалізована першою.

2. Scrum Master (Майстер Scrum).

Виступає як фасилітатор процесу Scrum. Не є керівником команди, а радше «служить» їй: допомагає усувати перешкоди, сприяє ефективній роботі. Слідкує, щоб команда дотримувалася правил Scrum, але при цьому не нав’язує рішень. Допомагає Product Owner у взаємодії з командою та зовнішніми стейкхолдерами. Навчає команду принципам самоорганізації, гнучкої роботи та ефективної комунікації.

Приклад: якщо під час спринту команда не може отримати доступ до тестового сервера, Scrum Master домовляється з іншими підрозділами й вирішує проблему.

3. Development Team (Команда розробки).

Кросфункціональна група професіоналів, які безпосередньо створюють продукт. У команді можуть бути програмісти, тестувальники, аналітики, дизайнери, DevOps-інженери. Scrum не встановлює чітких ролей усередині Development Team – вона сама визначає, хто і що робить. Команда є самоорганізованою: сама вирішує, як виконати завдання, які методи чи інструменти застосувати. Ідеальний розмір команди – 5-9 осіб: достатньо, щоб охопити всі компетенції, але не надто багато, щоб втратити ефективність комунікації.

Приклад: у спринті команда отримала завдання реалізувати «екран авторизації». Хтось із розробників працює над бекендом, інший – над фронтендом, дизайнер створює UI, тестувальник пише тести. Але вони всі відповідають колективно за кінцевий результат.

Взаємодія ролей у Scrum (рис.3.5):

✓ Product Owner визначає що треба робити (що створює найбільшу цінність).

✓ Development Team вирішує як це зробити технічно.

✓ Scrum Master допомагає і першим, і другим організувати процес, зняти перешкоди та забезпечити ефективну співпрацю.

Разом ці три ролі формують Scrum Team – ядро методології.

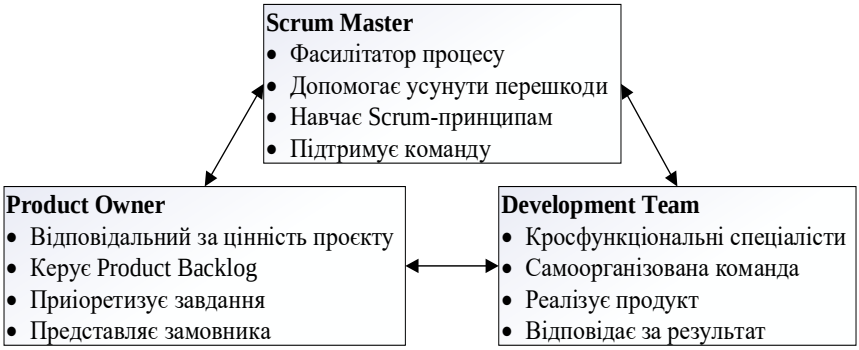

Рисунок 3.5 – Схема взаємодії ролей Scrum Team із поясненням функцій.

3.3.4. Артефакти Scrum

Артефакти у Scrum – це ключові інструменти управління інформацією, які допомагають усім учасникам бачити прозору картину проєкту та прогресу. Їх лише три, але вони критично важливі.

1. Product Backlog (Беклог продукту)

Це список усіх вимог, ідей, функцій, виправлень та покращень, які потенційно можуть увійти в продукт. Беклог постійно змінюється: щось додається, уточнюється, видаляється. Власник продукту (Product Owner) відповідає за його пріоритизацію – найбільш важливі та цінні завдання

стоять вище. Елементи беклогу часто описують у вигляді user stories («Як користувач я хочу… щоб…»).

Приклад: у мобільному банківському застосунку у беклозі можуть бути:

- ✓ «Як користувач, я хочу входити в застосунок за відбитком пальця»
- ✓ «Як клієнт, я хочу мати можливість робити перекази між картками»

2. Sprint Backlog (Беклог спринту)

Це підмножина Product Backlog, яку команда бере в роботу на конкретний спринт. Формується під час планування спринту: команда оцінює, скільки завдань реально можна виконати за 2-4 тижні.

У Sprint Backlog завжди є чіткі, досяжні цілі, які команда зобов'язується виконати. Протягом спринту він може уточнюватися, але в цілому повинен залишатися стабільним.

Приклад: якщо спринт триває 2 тижні, команда може взяти такі задачі:

- ✓ Реалізувати екран входу з біометрією
- ✓ Додати історію транзакцій
- ✓ Написати модульні тести для авторизації

3. Increment (Інкремент продукту)

Це результат роботи команди за спринт – готовий, працюючий шматок продукту, який можна показати замовнику. Кожен інкремент повинен відповідати стандартам якості (Definition of Done). Інкременти поступово накопичуються, утворюючи повний продукт.

Приклад: після першого спринту інкремент – екран входу з авторизацією.

- ✓ Після другого – вже доступний особистий кабінет із балансом.
- ✓ Після третього – додані перекази між картками.

Взаємозв'язок артефактів (рис.3.5).

1. Product Backlog = весь список ідей і вимог.
2. Sprint Backlog = частина беклогу, яку команда реалізує у поточному спринті.
3. Increment = готовий результат цього спринту.

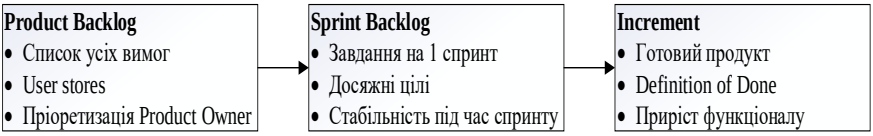

Рисунок 3.5 – Схема Scrum-артефакти (шлях від ідей до результату):

3.3.5. Події у Scrum (Scrum Events)

Scrum визначає чотири основні події (церемонії), які забезпечують прозорість, адаптивність і регулярний контроль прогресу. Кожна подія має чітку мету й часові рамки.

1. Sprint (Спринт).

Це основний цикл роботи у Scrum, що триває від 1 до 4 тижнів. Протягом спринту команда працює над завданнями зі Sprint Backlog.

Мета спринту – створити інкремент продукту, який можна продемонструвати замовнику.

Спринт починається з планування і закінчується оглядом та ретроспективою. У спринті забороняється змінювати цілі (але деталізація завдань може відбуватися).

2. Daily Scrum (Щоденна зустріч).

Коротка (15 хвилин) зустріч команди щоранку. Мета її – синхронізація та планування роботи на день. Кожен член команди відповідає на три питання:

- ✓ Що я зробив учора?
- ✓ Що планую зробити сьогодні?
- ✓ Які перешкоди заважають роботі?
- ✓ Це не звіт перед керівництвом, а інструмент командної координації.

3. Sprint Review (Огляд спринту).

Проводиться в кінці кожного спринту. Команда демонструє інкремент продукту замовнику та стейкхолдерам.

Мета – отримати зворотний зв’язок і зрозуміти, чи рухається команда в правильному напрямку. Може призвести до змін у Product Backlog.

4. Sprint Retrospective (Ретроспектива).

Завершальна подія після Sprint Review. Призначена лише для команди

(Product Owner, Scrum Master, Development Team).

Мета – оцінити роботу команди, виявити проблеми та обговорити, як покращити процес у наступному спринті. Ключові питання:

- ✓ Що було добре?
- ✓ Що можна покращити?
- ✓ Які конкретні дії ми зробимо в наступному спринті?

Загальний цикл подій (рис.3.6):

1. Планування спринту →
2. Daily Scrum (щоденні синхронізації) →
3. Sprint Review (демонстрація результатів) →
4. Retrospective (аналіз і покращення процесу)

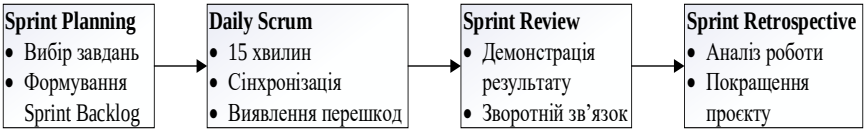

Рисунок 3.6 – Цикл спринту

Ця схема відображає замкнений цикл постійного вдосконалення та прозорої роботи команди.

3.3.6. Приклади застосування Scrum у IT-проєктах

1. Розробка мобільного застосунку банку

Ситуація: банк вирішує створити новий мобільний застосунок для управління рахунками.

Проблема: вимоги змінюються – регулятор оновлює правила безпеки, клієнти просять нові функції, конкуренти швидко впроваджують інновації.

Як допомагає Scrum:

- ✓ Product Owner формує беклог із вимогами.
- ✓ Команда працює короткими спринтами (2 тижні).
- ✓ Перший інкремент – базовий функціонал входу та перегляду балансу.
- ✓ Наступні спринти додають перекази між рахунками, push-

сповіщення, чат із підтримкою.

Результат: замість довгого очікування клієнти отримують продукт уже після кількох тижнів, і він поступово вдосконалюється.

2. SaaS-платформа для управління бізнесом

Ситуація: стартап створює хмарну систему для управління продажами.

Проблема: інвестори вимагають швидкого запуску мінімальної версії (MVP), щоб перевірити ідею на ринку.

Як допомагає Scrum:

✓ У беклозі є сотні можливих функцій, але Product Owner разом з командою вибирає лише найважливіші для MVP.

✓ Перший інкремент містить лише реєстрацію користувача та облік клієнтів.

✓ Подальші спринти додають інтеграцію з платіжними системами, аналітику, автоматизацію звітів.

Результат: стартап виходить на ринок уже за 2 місяці й отримує реальні відгуки користувачів.

3. Корпоративна CRM-система у великій компанії

Ситуація: велика міжнародна компанія хоче оновити свою CRM-систему для тисяч співробітників.

Проблема: різні відділи мають різні вимоги (продажі, маркетинг, підтримка клієнтів), які важко узгодити.

Як допомагає Scrum:

✓ Кожен відділ отримує представника, який співпрацює з Product Owner.

✓ У беклозі формується список функцій із пріоритетами.

✓ Команда поступово створює модулі: перший – управління контактами, другий – інтеграція з email, третій – панель аналітики.

Результат: кожен відділ отримує свої інструменти поетапно, і проект не зупиняється через бюрократію.

4. Розвиток ігор та геймдев

Ситуація: студія створює багатокористувацьку гру.

Проблема: ринок швидко змінюється, а гравці очікують частих оновлень.

Як допомагає Scrum:

- ✓ Спринти по 2 тижні дозволяють постійно випускати нові версії.
- ✓ У беклозі – завдання на нові карти, героїв, механіки.
- ✓ Щоразу після спринту випускається новий патч для гравців.

Результат: гра постійно розвивається, зростає залученість користувачів.

Висновок. Scrum ефективний у тих IT-проєктах, де:

- ✓ вимоги часто змінюються;
- ✓ важливо швидко показати результат користувачам;
- ✓ команда повинна працювати у тісній співпраці зі стейкхолдерами.

3.4. Kanban

Слово kanban (看板) японською означає «вивіска» або «картка». Виробнича система Kanban використовувала спеціальні картки, які передавалися між робітниками на конвеєрі й сигналізували, коли потрібно виготовити нову деталь або поповнити запаси.

3.4.1. Історія походження Kanban

Метод Kanban виник у 1950-х роках у компанії Toyota. Засновником концепції вважається інженер Тайіті Оно (Taiichi Ohno), один із творців системи Lean Production (бережливе виробництво). У післявоєнній Японії ресурси були обмежені, тому компанії шукали способи зменшення витрат і підвищення ефективності. Toyota розробила нову систему управління виробництвом, яка отримала назву Toyota Production System (TPS). Ця система дозволила уникати надвиробництва, зменшити складування і забезпечити принцип «just-in-time» (точно вчасно).

Приклад: якщо на складі залишалося 10 деталей, картка Kanban сигналізувала про необхідність виготовлення нової партії.

У 2000-х роках принципи Kanban були адаптовані для управління проєктами в IT. У 2004 році Девід Андерсон (David J. Anderson) представив Kanban як метод управління розробкою програмного забезпечення. Kanban став одним із популярних інструментів Agile-підходів, поряд із Scrum (рис.3.7).

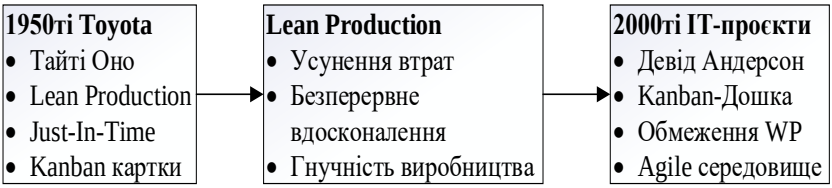

Рисунок 3.7 – Схема еволюції Kanban:

Основна ідея Kanban

✓ Візуалізація роботи за допомогою дошки Kanban, поділеної на колонки («To Do», «In Progress», «Done»).

✓ Обмеження кількості завдань, що перебувають у роботі (WIP limits).

✓ Безперервне вдосконалення процесу (Kaizen).

3.4.2. Kanban: загальний опис та принципи

Загальний опис Kanban

Kanban – це гнучкий метод управління робочими процесами, що дозволяє командам ефективно організовувати виконання завдань. Його ключова ідея – візуалізація процесу за допомогою Kanban-дошки, де всі завдання відображаються у вигляді карток, розміщених у відповідних колонках («To Do», «In Progress», «Done»).

Такий підхід робить роботу прозорою для всієї команди, допомагає виявити «вузькі місця» й забезпечує безперервний потік виконання завдань.

Kanban часто порівнюють із Scrum:

✓ Scrum працює ітераціями (спринтами),

✓ Kanban – безперервним потоком роботи без жорстких часових рамок.

Kanban спирається на три основні принципи:

1. Візуалізація роботи.

Усі завдання відображаються на дошці Kanban. Це дозволяє кожному учаснику команди бачити, що відбувається в проєкті. Використання кольорових карток допомагає швидко ідентифікувати типи завдань (баги, нові фічі, документація тощо).

2. Обмеження кількості завдань у роботі (WIP limits – Work In

Progress).

Головна ідея Kanban полягає у тому, що не можна «тягнути» в роботу безліч завдань одночасно. Наприклад, у колонці «In Progress» може бути максимум 3 завдання. Якщо хтось хоче взяти нове – спочатку треба завершити одне із тих, що вже виконується. Це зменшує багатозадачність і підвищує якість результатів.

3. Безперервне вдосконалення (Kaizen)

Kanban не передбачає жорстких ролей і подій (як у Scrum), але акцентує увагу на постійних покращеннях. Команда аналізує процес, шукає способи усунення «вузьких місць» і оптимізації роботи. Ідея Kaizen («зміни на краще») стала частиною культури багатьох IT-команд.

3.4.3. Візуалізація роботи: Kanban-дошка та карти завдань

Основний інструмент Kanban – це дошка (Kanban board, рис.3.8).

Рисунок 3.8 – Схема прикладу Kanban-дошки з WIP-лімітами (наприклад, 3 колонки: «To Do», «In Progress» (макс. 3 задачі), «Done»).

Вона ділиться на колонки, які відображають етапи робочого процесу. Найпростіший варіант: To Do (завдання, які треба виконати), In Progress (завдання у процесі роботи), Done (завершені завдання). У складніших процесах додають додаткові колонки, наприклад: «Code Review», «Testing», «Deployment». Таким чином Kanban-дошка відображає поточний статус усіх завдань і дозволяє команді бачити прогрес.

Кожне завдання представлено у вигляді Карти завдань (Kanban cards). Картка містить базову інформацію:

- ✓ короткий опис завдання;
- ✓ відповідального виконавця;
- ✓ пріоритет;

- ✓ терміни виконання;
- ✓ додаткові мітки (наприклад, «Bug», «Feature», «High Priority»).

У сучасних цифрових інструментах (Jira, Trello, Azure DevOps) картки інтерактивні:

- ✓ можна додавати коментарі;
- ✓ прикріплювати файли;
- ✓ відстежувати час виконання;
- ✓ змінювати статус перетягуванням між колонками.

Навіщо потрібна візуалізація:

- ✓ Прозорість: вся команда бачить, над чим працює кожен.
- ✓ Виявлення «вузьких місць» у процесі (наприклад, забагато завдань у «Testing»).
- ✓ Легше керувати пріоритетами і ресурсами.
- ✓ Мотивація: команда бачить поступове зростання колонки «Done».

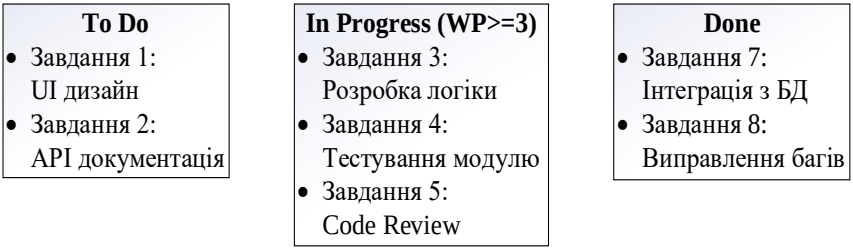

Рисунок 3.9 – Схема прикладу Kanban-дошки з картами завдань (наприклад: «To Do» – 2 задачі, «In Progress» – 3 задачі, «Done» – 2 задачі).

Існує безліч чудових додатків для створення Kanban-дошок, які підійдуть для різних потреб – від особистого планування до великих командних проєктів. У таблицях 3.2-3.3 наведені найпопулярніші та найефективніші з них, які можна використовувати як студентам, так і професіоналам:

Таблиця 3.2 – Додатки для створення Kanban-дошок (частина 1)

Характеристика	Trello	Jira Software	Asana
Основне призначення	Простий Kanban, особисті та малі команди	Agile-розробка, Scrum/Kanban, DevOps	Управління проєктами, командна співпраця
Складність вивчення	Низька	Висока (для повного функціоналу)	Середня
Гнучкість	Середня	Висока (налаштування робочих процесів)	Висока
Безкоштовний тариф	Так (з обмеженнями)	Так (до 10 користувачів, з обмеж.)	Так (з обмеженнями)
Інтеграції	Багато (через "Power-Ups")	Дуже багато (екосистема Atlassian)	Багато
Основні сильні сторони	Простота, візуальна інтуїтивність, швидкий старт	Потужність для розробки, звіти, CI/CD	Гнучкість, керування завданнями, цілі
Основні недоліки	Обмежений для великих, складних проєктів	Крива навчання, може бути дорогим	Для складних проєктів може бракувати потужності
Для студентів КІ	Дуже добре (для малих проєктів)	Дуже добре (якщо вивчаєте Agile/Scrum)	Добре

Таблиця 3.3 – Додатки для створення Kanban-дошок (частина 2)

Характеристика	Monday.com	Notion	ClickUp
Основне призначення	Управління робочим процесом (Work OS), візуальні дошки	Універсальний робочий простір, нотатки, бази даних	Управління проєктами "все в одному", продуктивність
Складність вивчення	Середня	Середня/Висока (через гнучкість)	Середня/Висока (через кількість функцій)
Гнучкість	Дуже висока (візуальні налаштування)	Дуже висока (як конструктор)	Дуже висока
Безкоштовний тариф	Ні (лише пробний період)	Так (з щедрими обмеженнями)	Так (з щедрими обмеженнями)
Інтеграції	Багато	Середня (постійно зростає)	Дуже багато
Основні сильні сторони	Візуалізація, кастомізація, автоматизація	Все в одному, бази даних, персоналізація	Багатофункціональність, кастомізація
Основні недоліки	Ціна, може здаватися надто яскравим	Крива навчання, не спеціалізований Kanban	Може бути перевантаженим функціями
Для студентів КІ	Добре	Відмінно (якщо ви хочете гнучкість)	Дуже добре

3.4.4. Обмеження WIP (Work In Progress)

WIP (Work In Progress) – це кількість завдань, які перебувають у роботі на певному етапі процесу. В Kanban важливо встановлювати ліміти WIP, тобто визначати максимальну кількість задач, які можуть одночасно виконуватись у колонці «In Progress».

Навіщо потрібні WIP-ліміти

1. Зменшення багатозадачності

✓ Якщо команда бере на себе забагато завдань, продуктивність падає.

✓ Ліміти змушують завершувати вже розпочаті задачі перед тим, як брати нові.

2. Фокус на якості, а не кількості

✓ Завдання виконуються більш ретельно.

✓ Менше помилок та багів, бо виконавець зосереджений на одному завданні.

3. Виявлення «вузьких місць»

✓ Якщо якась колонка перевищує ліміт WIP, це означає, що там накопичуються задачі.

✓ Це сигнал, що саме на цьому етапі процесу є проблеми (наприклад, тестування занадто повільне).

✓ Візуалізація WIP-лімітів. На Kanban-дошці часто прямо вказується обмеження (наприклад: «In Progress (≤ 3)»). Це допомагає швидко зрозуміти, чи дотримується команда лімітів.

Приклад. У команді є правило: в колонці «In Progress» може бути максимум 3 завдання. Якщо вже є три задачі, нові додавати не можна, доки хоча б одна не буде завершена. Це стимулює команду завершувати поточну роботу, а не брати нові задачі (рис.3.10).

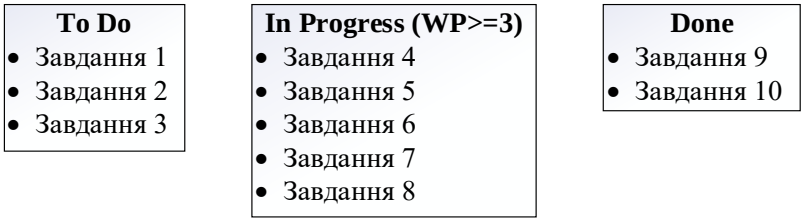

Рисунок 3.10 – Приклад Kanban-дошки з WIP-лімітом (у колонці «In Progress» дозволено максимум 3 задачі, але їх стало 5 – візуалізація перевантаження).

3.4.5. Безперервне вдосконалення (Kaizen)

Слово Kaizen (яп. 改善) перекладається як «зміни на краще». Цей підхід був сформований у Японії після Другої світової війни та став основою філософії бережливого виробництва (Lean Production).

У контексті управління проєктами Kaizen означає постійне покращення процесів, навіть маленькими кроками, що в довгостроковій перспективі дає значні результати (рис.3.11).

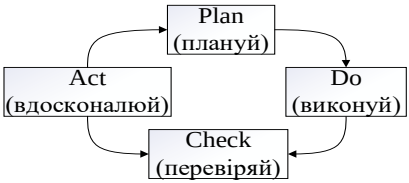

Рисунок 3.11 – Схема циклу Kaizen:
«Плануй → Виконуй → Перевіряй → Вдосконалюй (PDCA cycle)».

Kanban не є жорсткою методологією, він більше нагадує гнучку систему управління. Тому однією з ключових ідей Kanban є безперервне вдосконалення процесу роботи команди. Це відбувається через:

- ✓ аналіз продуктивності;
- ✓ пошук «вузьких місць»;
- ✓ покращення комунікацій;
- ✓ оптимізацію робочих процесів.

Інструменти безперервного вдосконалення

1. Візуалізація метрик

✓ Використання діаграм (наприклад, Cumulative Flow Diagram – діаграма накопиченого потоку) для аналізу швидкості роботи.

✓ Відстеження часу виконання завдань (Cycle Time).

2. Регулярні обговорення

✓ Команда проводить мітинги для обговорення проблем і пропозицій.

✓ Це може бути аналог Scrum Retrospective, але у більш вільному форматі.

3. Малі зміни, але постійно

✓ Замість великих реформ – маленькі поліпшення щодня.

✓ Наприклад: оптимізація шаблонів карток, уточнення критеріїв «готовності», покращення комунікації.

Приклад у IT-проєкті. Команда тестувальників помітила, що завдання часто «застрягають» у колонці «Testing». Після аналізу виявилось, що тестувальники витрачають багато часу на ручні перевірки. Було впроваджено автоматизовані тести, що дозволило зменшити час тестування та збільшити пропускну здатність команди. Це класичний приклад Kaizen у дії.

3.4.6. Приклади застосування Kanban у IT-проєктах

1. Розробка вебзастосунків.

Команда фронтенд-розробників використовує Kanban-дошку з колонками: «Backlog», «Design», «Development», «Testing», «Done». WIP-ліміти встановлені для «Development» і «Testing», щоб уникнути перевантаження. Це дозволяє швидко реагувати на нові запити від клієнтів і зберігати прозорість процесу.

2. Підтримка та обслуговування ПЗ.

Kanban особливо ефективний для операційних команд підтримки (support, maintenance). Наприклад, у службі підтримки користувачів заявки надходять безперервно. Завдяки Kanban команда може пріоритезувати задачі («Critical», «Major», «Minor») і відслідковувати їх статус у реальному часі.

3. DevOps та інфраструктура.

Команди DevOps застосовують Kanban для керування CI/CD-процесами. - Завдання типу «Налаштувати пайплайн», «Оновити контейнер», «Впровадити моніторинг» відображаються на Kanban-дошці. Це допомагає координувати роботу між розробниками, тестувальниками та адміністраторами.

4. Геймдев (ігрова індустрія).

У розробці ігор Kanban часто поєднується зі Scrum (ScrumBan). Наприклад, художники, аніматори та звукорежисери працюють за Kanban, бо їхня робота більше орієнтована на безперервний потік, а програмісти – за Scrum. Це дозволяє узгоджувати креативні та технічні процеси.

5. Стартапи

У невеликих стартапах Kanban часто використовується замість складніших фреймворків. Він дозволяє швидко візуалізувати поточні завдання, адаптувати процес без великих витрат часу на планування.

```
Web Development:
      Backlog –• Design –• Dev –• Test –• Done
Support & Maintenance:
      Пріоритезація заявок (Critlcal/Major/Mlnor)
DevOps:
      CI/CD завдання: пайплайни, контейнери, моніторинг
GameDev:
      ScrumBan: креатив через Kanban, код через Scrum
Startups:
      Швидка адаптація без складних процесів
```

Рисунок 3.12 – Приклади застосування Kanban у різних типах ІТ-команд (Web Dev, Support, DevOps, GameDev, Startups).

3.5. Порівняння Scrum і Kanban

Таблиця 3.4 містить загальне порівняння відмінностей Scrum і Kanban

Таблиця 3.4 – Відмінності Scrum і Kanban

Критерій	Scrum	Kanban
Структура	Чітко регламентований фреймворк	Гнучка система, немає жорстких правил
Ітерації	Робота у фіксованих спринтах (2-4 тижні)	Потік завдань без фіксованих ітерацій
Ролі	Product Owner, Scrum Master, Development Team	Ролі не визначені, лише команда
Події	Sprint Planning, Daily Scrum, Review, Retrospective	Подій немає, лише обговорення за потреби
WIP-ліміти	Не регламентовані, але є завантаженість спринту	Основний механізм управління процесом
Придатність	Коли є потреба у прогнозуваності та структурі	Коли потрібно працювати безперервним потоком

Схожості Scrum і Kanban:

1. Обидві методології належать до Agile-підходів.
2. Основна ідея – гнучкість та постійна адаптація до змін.
3. Використовують візуалізацію роботи (Scrum-дошка, Kanban-дошка).
4. Орієнтовані на інкрементальну поставку цінності користувачу.
5. Сприяють прозорості процесу та командній взаємодії.

Коли обрати Scrum, а коли Kanban:

- ✓ Scrum підходить, якщо:
  - ✓ команда нова і потребує чіткої структури;
  - ✓ проєкт потребує регулярних інкрементів;
  - ✓ замовнику важливі передбачувані результати кожні 2-3 тижні.
- ✓ Kanban підходить, якщо:
  - ✓ робота надходить у безперервному потоці (підтримка, DevOps);
  - ✓ важлива гнучкість і відсутність жорстких рамок;
  - ✓ команда досвідчена і самостійно організовує процес

3.5.1. Вибір підходу залежно від типу проєкту

1. Тип проєкту: стартап

- ✓ Проблема: висока невизначеність, часті зміни вимог.
- ✓ Кращий підхід: Kanban.
- ✓ Дає максимальну гнучкість.
- ✓ Не потребує тривалого планування.
- ✓ Легко масштабувати і змінювати процес «на льоту».

2. Комерційні IT-продукти з регулярними релізами (рис.3.12)

- ✓ Проблема: потрібна передбачуваність і чіткі релізні цикли.
- ✓ Кращий підхід: Scrum.
- ✓ Дає структуру через спринти.
- ✓ Регулярні релізи (кожні 2-4 тижні).
- ✓ Добре працює, коли замовник очікує демонстрації прогресу.

3. Підтримка існуючих систем

- ✓ Проблема: постійний потік заявок від користувачів.
- ✓ Кращий підхід: Kanban.

✓ Оптимальний для роботи з багатьма дрібними завданнями.

✓ Дозволяє пріоритезувати критичні заявки.

✓ Зменшує час реакції на інциденти.

4. Великі корпоративні проєкти

✓ Проблема: кілька команд, висока складність, потреба в синхронізації.

✓ Кращий підхід: Scrum або ScrumBan.

✓ Scrum дає структуру для синхронізації між командами.

✓ Можливе поєднання зі ScrumBan для підрозділів із більш гнучкими процесами.

5. Дослідницькі та інноваційні проєкти (R&D)

✓ Проблема: невизначеність результатів, багато експериментів.

✓ Кращий підхід: Kanban.

✓ Дозволяє швидко відкидати непрацюючі ідеї.

✓ Мінімізує бюрократію.

Рисунок 3.12 – Схема «Вибір Scrum чи Kanban залежно від типу проєкту»

3.5.2. Гібридні моделі (Scrumban)

Scrumban – це гібрид Scrum і Kanban, який поєднує структурованість Scrum із гнучкістю Kanban. Спочатку був запропонований як спосіб перехідної моделі для команд, які почали працювати за Scrum, але хотіли більшої гнучкості. Сьогодні Scrumban застосовується як самостійний підхід.

Основні особливості

1. Ітеративність від Scrum

✓ Команда може працювати в спринтах (наприклад, 2 тижні), але планування гнучкіше.

✓ Замість жорсткого Sprint Backlog – динамічний набір завдань.

2. Потоковість від Kanban

✓ Використання Kanban-дошки для відображення завдань.

✓ Встановлення WIP-лімітів.

✓ Завдання можуть додаватися навіть під час спринту.

3. Фокус на адаптацію

✓ Планування не є суворо фіксованим.

✓ Команда може змінювати обсяг робіт у спринті за потреби.

Коли застосовують Scrumban

✓ Великі компанії, які прагнуть поєднати передбачуваність Scrum із гнучкістю Kanban.

✓ Проєкти з постійним потоком завдань, але потрібні регулярні точки контролю (наприклад, релізи кожні кілька тижнів).

✓ Команди підтримки або DevOps, яким треба поєднати операційні завдання з елементами довгострокового планування.

Приклад у IT-проєкті

Команда працює над SaaS-продуктом. Вони проводять 2-тижневі спринти, але на Kanban-дошці відображають усі завдання (розробка нових фіч, виправлення багів, підтримка). Якщо з'являється критичний баг – його можна додати прямо під час спринту. Таким чином, команда має і регулярність релізів, і гнучкість Kanban (рис.3.13).

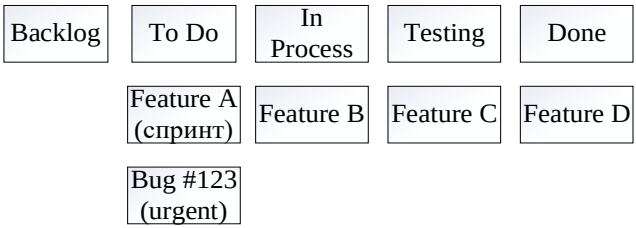

Рисунок 3.13 – Гібридна дошка Scrumban:
Стовпці від Kanban: Backlog → To Do → In Progress → Testing → Done.
Спринтове планування від Scrum (мітки завдань належать до поточного спринту).

3.6. Переваги та виклики Agile-підходів

Гнучкі підходи мають певні виклики та переваги над каскадними.

3.6.1. Ризики та проблеми: масштабування, роль замовника, зрілість команди

1. Масштабування Agile

Agile чудово працює у малих командах (5-9 осіб), де всі учасники тісно взаємодіють. Однак при спробі масштабування на десятки чи сотні людей виникають проблеми:

✓ Координація між командами. Декілька Scrum-команд можуть мати залежності у роботі, що уповільнює процес.

✓ Розмиття ролей. Product Owner не завжди може ефективно керувати кількома командами.

✓ Втрата прозорості. При великій кількості завдань складно зберегти повну видимість прогресу.

Для вирішення цих проблем розроблено Scaled Agile Framework (SAFe), LeSS (Large-Scale Scrum), Disciplined Agile Delivery (DAD), які намагаються зберегти гнучкість, але вводять додаткову структуру.

2. Роль замовника

Agile передбачає активну участь замовника (Customer / Product Owner) у процесі:

✓ він має постійно формувати пріоритети;

✓ брати участь у регулярних зустрічах;

✓ перевіряти інкременти продукту;

✓ швидко надавати зворотний зв'язок.

Ризики:

✓ Замовник може бути занадто зайнятий і не приділяти достатньо уваги команді.

✓ У великих організаціях замовник часто діє через посередників, що знижує ефективність.

✓ Непослідовність у прийнятті рішень призводить до хаотичності в беклозі.

3. Зрілість команди

Agile найбільш ефективний, коли команда самоорганізована і має високий рівень зрілості. Але на практиці зустрічаються проблеми:

✓ Низький рівень компетенцій. Молоді або нові фахівці можуть не мати досвіду самостійного прийняття рішень.

✓ Відсутність довіри між членами команди уповільнює роботу.

✓ Небажання брати відповідальність за продукт (чекання інструкцій від менеджера).

У таких випадках Agile може перетворитися на хаос – коли немає ані жорсткої структури, ані справжньої самоорганізації.

Висновок. Agile не є «чарівною паличкою». Його ефективність залежить від (рис.3.14):

✓ масштабу проєкту;

✓ активності й зрілості замовника;

✓ досвіду та відповідальності команди.

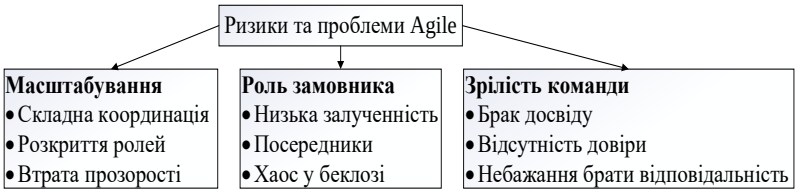

Рисунок 3.14 – Основні ризики Agile

3.6.2. Переваги Agile

1. Гнучкість і адаптивність.

Agile дозволяє швидко реагувати на зміни вимог, ринку чи пріоритетів замовника. Класичні моделі (наприклад, Waterfall) жорстко прив'язані до початкового плану, тоді як Agile дає можливість коригувати продукт навіть на пізніх етапах розробки. Це особливо важливо для динамічних IT-ринків, де вимоги користувачів змінюються постійно.

2. Орієнтація на клієнта та користувача.

Agile передбачає тривалу співпрацю із замовником і регулярну перевірку інкрементів продукту. Замовник отримує проміжні результати через кожен спринт і може оцінити, чи відповідає продукт його очікуванням.

Це мінімізує ризик, що кінцевий результат не задовольнить потреби ринку.

3. Прозорість і контроль.

Agile створює високу прозорість процесів через Kanban- чи Scrum-дошки, щоденні мітинги, ретроспективи. Замовник і команда завжди знають, який прогрес зроблено, які завдання в роботі, які проблеми виникли. Це полегшує управління ризиками.

4. Якість продукту.

Завдяки частим перевіркам ітерацій виявлення дефектів відбувається на ранніх етапах. Команда може швидко виправити баги й покращити функціонал. Безперервне тестування (Continuous Integration / Continuous Testing) робить продукт стабільнішим.

5. Мотивація та ефективність команди.

Agile робить команду самоорганізованою, дає їй більше свободи у прийнятті рішень. Це підвищує відповідальність, залученість і мотивацію учасників. Регулярні ретроспективи сприяють постійному вдосконаленню як продукту, так і процесу роботи.

Висновок. Agile дає бізнесу швидкість і конкурентні переваги, замовнику – прозорість і можливість впливати на продукт, а команді – мотивацію та розвиток. Це і пояснює, чому Agile став домінуючим підходом у сучасному ІТ (рис.3.15).

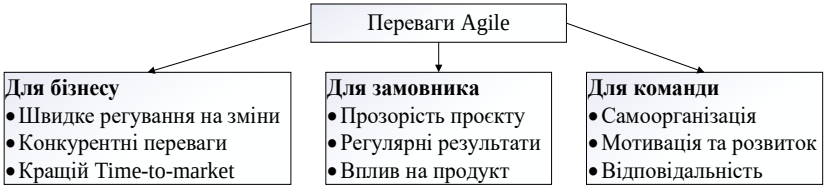

Рисунок 3.15 – Переваги Agile для бізнесу, замовника і команди

3.7. Практичні кейси застосування Agile

Розглянемо декілька прикладів застосування гучних підходів для управляння Іт-проєктами.

3.7.1. Приклади у стартапах, продуктових компаніях, аутсорсингових командах.

1. Стартапи

Ситуація: стартап створює мобільний застосунок для доставки їжі. Вимоги ринку швидко змінюються: користувачі очікують інтеграцію з Apple Pay, push-сповіщення, систему лояльності.

Виклик: неможливо одразу точно передбачити весь функціонал продукту.

Рішення (Agile):

- ✓ застосування Scrum із двотижневими спринтами;
- ✓ регулярні демо для інвесторів і тестових користувачів;
- ✓ беклог постійно коригується залежно від реакції ринку.

Результат: стартап швидко випускає MVP, отримує відгуки, знаходить інвесторів і масштабуватиме продукт далі.

2. Продуктові компанії

Ситуація: середня IT-компанія розробляє SaaS-платформу для управління фінансами малого бізнесу.

Виклик: потрібно регулярно випускати нові функції, залишаючись конкурентними.

Рішення (Agile):

- ✓ використання Kanban для роботи над постійним потоком задач;
- ✓ Scrum для основних релізних функцій;
- ✓ впровадження CI/CD (безперервна інтеграція та доставка).

Результат: компанія може щотижня додавати нові можливості та швидко виправляти баги, отримуючи позитивний фідбек від клієнтів.

3. Аутсорсингові команди

Ситуація: українська компанія-розробник виконує замовлення для європейського банку на створення внутрішньої CRM-системи.

Виклик: замовник у Європі змінює вимоги в процесі роботи, є часові відмінності та культурні бар’єри.

Рішення (Agile):

- ✓ використання Scrum з регулярними онлайн-демо та щотижневими зустрічами зі стейкхолдерами;

✓ прозора Kanban-дошка (Jira, Trello) для клієнта;

✓ акцент на ролі Product Owner як посередника між клієнтом і командою.

Результат: замовник має прозорість і контроль, команда – чіткі пріоритети, продукт успішно розгорнуто в банку.

Висновок. Agile підходи добре працюють у різних контекстах:

✓ у стартапах дають швидкий вихід на ринок;

✓ у продуктових компаніях – стабільний розвиток і оновлення;

✓ в аутсорсингу – ефективну співпрацю з клієнтом.

3.7.2. Порівняння результатів Agile vs Waterfall на реальних прикладах

1. Розробка мобільного застосунку (стартап vs корпорація)

✓ Waterfall: Корпорація планувала мобільний банківський застосунок. Усі вимоги були зафіксовані на початку. Після 18 місяців розробки продукт вийшов на ринок, але користувачі вже очікували інший функціонал (біометрія, push-сповіщення), чого в ТЗ не було. Результат – низька популярність і необхідність термінового редизайну.

✓ Agile: Стартап із доставки їжі використовував Scrum із 2-тижневими спринтами. MVP запустили через 3 місяці, отримали відгуки, додали функції оплати через Apple Pay. Продукт швидко набрав аудиторію та інвестиції.

Висновок: Agile краще підходить у швидкозмінному середовищі.

2. Державні проєкти (податкова система)

✓ Waterfall: В одній країні розробляли електронну систему податкової звітності. Через суворі вимоги законодавства та регламентів обрали каскадну модель. Після кількох років проєкт завершився – і система працювала стабільно, адже всі юридичні нюанси було враховано ще на старті.

✓ Agile: Схожий проєкт в іншій країні розроблявся гнучкими ітераціями. Через часті зміни законів команда встигала вносити корективи до кожного релізу. Це дозволило швидко адаптувати систему до нових вимог, але проєкт був дорожчим і складнішим у координації.

Висновок: Waterfall краще у випадках, де потрібна юридична стабільність, Agile – там, де зміни відбуваються регулярно.

3. Ігрова індустрія (AAA vs інді-студія)

✓ Waterfall: AAA-студія планувала велику гру на кілька років. Вимоги й дизайн були визначені на старті. У підсумку випустили якісний продукт, але без урахування актуальних трендів (наприклад, battle royale).

✓ Agile: Інді-студія створила гру в стилі survival. Використовували Kanban, релізили нові версії щотижня, слухали ком'юніті. Гра поступово «виросла» до хіта на Steam.

Висновок: Agile дає інді-командам перевагу у швидкому реагуванні на ринок.

Загальний висновок (табл.3.5):

✓ Agile – швидкість, адаптивність, гнучкість у змінному середовищі.

✓ Waterfall – стабільність, чіткість, передбачуваність у регульованих сферах.

Таблиця 3.5 – Порівняльна Agile vs Waterfall на прикладах (Стартап, Державний проєкт, Ігри).

Сфера	Waterfall	Agile
Стартап (мобільний застосунок)	18 міс. розробки. продукт застарів до релізу	MVP через 3 міс,, швидкий фідбек, зростання аудиторії
Державний проєкт (податкова система)	Стабільна система після кількох років, враховані всі законодавчі акти	Регулярні релізи з адаптацією до законів, але дороже
Ігрова індустрія	AAA-гра з довгим циклом, без врахування нових трендів	Інді-гра з тижневими оновленнями, стала хітом

3.8. Загальні висновки

3.8.1. Ключові підсумки

1. Agile як філософія управління

✓ Agile Manifesto (4 цінності, 12 принципів) → основа сучасного підходу до розробки.

✓ Agile – це не лише набір методик, а насамперед мислення,

орієнтоване на зміни та людей.

2. Scrum і Kanban як практичні інструменти

✓ Scrum: ітерації, ролі (Product Owner, Scrum Master, Development Team), артефакти та церемонії.

✓ Kanban: візуалізація завдань, WIP-обмеження, Kaizen.

3. Де Agile ефективний, а де краще класичні методи

✓ Agile = стартапи, продуктові компанії, аутсорсинг, ігрова індустрія.

✓ Waterfall/PMBOK = держпроєкти, військові розробки, великі системи з регуляторними обмеженнями.

3.8.2. Питання для самоперевірки

1. Які 4 основні цінності та 12 принципів Agile?
2. Чим Agile відрізняється від Waterfall у плануванні та гнучкості?
3. Назвіть ключові ролі у Scrum і їхню відповідальність.
4. Що таке Scrum-артефакти (Product Backlog, Sprint Backlog, Increment)?
5. У чому суть щоденного Scrum (Daily Scrum)?
6. Які основні елементи Kanban-дошки?
7. Для чого встановлюються WIP-обмеження?
8. Що означає поняття Kaizen?
9. Наведіть приклади сфер, де Agile підходить краще за Waterfall.
10. Які ризики та виклики існують у впровадженні Agile?

3.8.3. Завдання для самостійної роботи

1. Побудувати приклад Kanban-дошки для команди, що працює над університетським проєктом (наприклад, створення сайту кафедри).
2. Скласти Scrum-беклог для розробки мобільного застосунку (мінімум 10 user stories).
3. Порівняти Agile і Waterfall у контексті розробки:

✓ (а) мобільної гри,

✓ (б) державної системи електронного документообігу.

✓ Написати короткий висновок (1-2 сторінки).

3.9. Контрольні запитання

1. Чому традиційні методи управління проєктами виявилися недостатньо ефективними для сучасної IT-галузі?
2. Що таке Agile? Назвіть основні цінності та принципи Agile-маніфесту.
3. Які проблеми традиційних моделей вирішує підхід Agile? Наведіть конкретні приклади.
4. Що таке Scrum? Опишіть основні ролі в Scrum-команді та їх відповідальність.
5. Що являє собою спринт у Scrum? Яка типова тривалість і яка мета кожного спринту?
6. Охарактеризуйте основні артефакти Scrum: Product Backlog, Sprint Backlog, Increment.
7. Які церемонії (події) передбачені у Scrum? Поясніть призначення кожної.
8. Що таке Kanban? У чому полягають ключові принципи Kanban-підходу?
9. Як влаштована Kanban-дошка і які типові статуси завдань вона відображає?
10. У чому відмінність між Scrum і Kanban? Коли доцільно застосовувати кожен із них?
11. Що таке WIP-ліміти у Kanban і чому вони важливі для управління потоком завдань?
12. Як порівнюються Agile-методології з класичним Waterfall за критеріями гнучкості, ризиків і залучення клієнта?
13. Що таке velocity у Scrum і як цей показник використовується для планування?
14. Яку роль відіграє Product Owner у Scrum? Яка різниця між Product Owner і проєктним менеджером?
15. Наведіть приклад реального IT-проєкту, де застосування Agile дало відчутні переваги. Обґрунтуйте вибір методології.

4. БІЗНЕС-АНАЛІЗ В ІТ-ПРОЄКТАХ

4.1. Вступ. Актуальність бізнес-аналізу в ІТ

У сучасних ІТ-проєктах бізнес-аналіз є однією з ключових дисциплін, що визначає успіх чи провал розробки. За даними Standish Group (звіт CHAOS Report), понад 40 % проєктів провалюються або не приносять очікуваної цінності саме через нечітко визначені або неправильно сформульовані вимоги. У такій ситуації бізнес-аналітик (Business Analyst, BA) стає тією ланкою, яка з'єднує бізнес-цілі замовника з технічними можливостями команди розробки.

Чому бізнес-аналіз настільки важливий?

✓ Забезпечення розуміння між бізнесом і технічною командою.

Часто замовник описує вимоги у бізнес-термінах («система має підвищити лояльність клієнтів»), а команда мислить у технічних категоріях («реалізувати push-нотифікації»). Бізнес-аналітик перекладає одну мову в іншу.

✓ Формалізація вимог.

Неоформлені ідеї та побажання потребують структурування. Без чіткої документації команда ризикує створити продукт, що не відповідає очікуванням користувачів.

✓ Зниження ризиків.

Вчасно виявлені протиріччя або прогалини у вимогах дозволяють уникнути дорогих переробок на пізніх етапах (ефект «снігової кулі»).

✓ Оптимізація ресурсів.

Якщо команда витрачає час на «непотрібні» функції, це означає втрату бюджету. Аналітик допомагає визначити, що дійсно створює цінність для бізнесу.

Місце бізнес-аналізу серед інших дисциплін проєктного менеджменту:

✓ У Waterfall бізнес-аналіз концентрується на ретельному зборі й документуванні вимог ще до початку розробки.

✓ У Agile бізнес-аналітик виступає в ролі партнера Product Owner, допомагає формувати беклог, створює user stories та підтримує постійний контакт із замовником.

Таким чином, бізнес-аналіз – це міст між баченням і реалізацією, без якого будь-який проєкт ризикує втратити напрямок (рис.4.1).

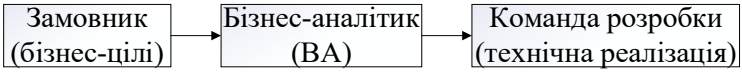

Рисунок 4.1 – Місце бізнес-аналітика в IT-проєкті.

4.2. Роль бізнес-аналітика

Бізнес-аналітик (Business Analyst, BA) – це спеціаліст, який виступає посередником між бізнес-замовником і технічною командою. Його головна місія – зрозуміти справжні потреби бізнесу й допомогти команді розробників реалізувати їх у вигляді робочого продукту.

Основні завдання бізнес-аналітика (рис.4.2):

1. Збір вимог:
- ✓ Проведення інтерв'ю та воркшопів із замовниками.
- ✓ Аналіз бізнес-процесів і виявлення проблем.
- ✓ Формалізація бізнес-потреб у конкретні вимоги.

2. Аналіз і моделювання:
- ✓ Побудова діаграм бізнес-процесів (BPMN, UML).
- ✓ Виявлення залежностей між функціональними та нефункціональними вимогами.
- ✓ Пріоритизація вимог (наприклад, за моделлю MoSCoW).

3. Документування:
- ✓ Створення BRD (Business Requirements Document).
- ✓ Підтримка живого беклогу в Agile.
- ✓ Ведення матриці трасування вимог.

4. Комунікація:
- ✓ Постійна взаємодія з усіма стейкхолдерами.
- ✓ Фасилітація зустрічей між бізнесом і командою.
- ✓ Усунення непорозумінь між технічними та бізнес-термінами.

5. Підтримка команди:
- ✓ Уточнення вимог у процесі розробки.
- ✓ Допомога QA у створенні тест-кейсів.

✓ Аналіз впливу змін у вимогах на бюджет і терміни.

Ключові компетенції бізнес-аналітика:

✓ Soft skills: активне слухання, комунікація, фасилітація, критичне мислення, вміння ставити правильні питання.

✓ Hard skills: моделювання (BPMN, UML), SQL, створення прототипів, володіння інструментами (Jira, Confluence, Figma, Enterprise Architect).

Роль BA у різних типах проєктів:

✓ У стартапі – бізнес-аналітик часто поєднує ролі Product Owner, UX-дизайнера та тестувальника.

✓ У великій корпорації – BA має вузьку спеціалізацію, працює у зв'язці з системними аналітиками та архітекторами.

✓ В аутсорсингу – виступає основним каналом комунікації з клієнтом і «перекладає» бізнес-вимоги у завдання для команди.

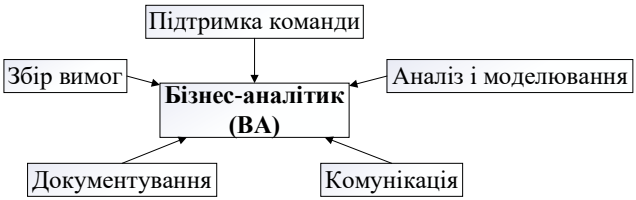

Рисунок 4.2 – Функції бізнес-аналітика в проєкті»,

4.2.1. Визначення бізнес-аналітика як «посередника» між бізнесом і технічною командою

Бізнес-аналітик (Business Analyst, BA) – це фахівець, який виступає посередником між стейкхолдерами бізнесу (замовниками, користувачами, керівниками) та технічною командою (розробниками, тестувальниками, архітекторами). Його головна роль полягає у тому, щоб:

✓ зрозуміти справжні потреби бізнесу, які іноді не завжди чітко сформульовані;

✓ перекласти ці потреби у зрозумілі технічні вимоги;

✓ забезпечити взаєморозуміння між сторонами, які часто мислять різними «мовами».

Чому потрібен посередник?

1. Бізнес говорить своєю мовою.

Замовники формулюють потреби у вигляді цілей: «покращити продажі», «зменшити витрати», «збільшити лояльність клієнтів». Для розробників це звучить занадто загально і не містить конкретних завдань.

2. Команда розробки мислить технічно.

Програмісти думають у категоріях технологій: бази даних, API, архітектурні патерни. Замовник часто не розуміє, що стоїть за цими термінами.

3. Ризик непорозумінь.

Без BA бізнес-очікування і результат роботи команди можуть радикально відрізнятися. Класичний приклад – «гойдалка» (swing illustration): те, що хотів замовник, що описав менеджер, як зрозуміла команда, і що зрештою було реалізовано (рис.4.3).

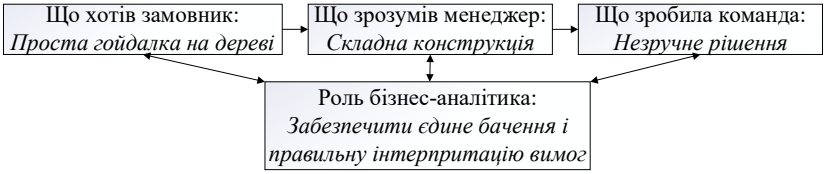

Рисунок 4.3 – Схема “гойдалки”
(«What the customer wanted vs what was built»).

Мета бізнес-аналітика:

✓ Зробити так, щоб кінцевий продукт відповідав реальним потребам користувачів, а не тільки формальній специфікації.

✓ Забезпечити узгодженість між різними групами зацікавлених сторін.

✓ Створити умови, коли усі сторони однаково розуміють цілі проєкту.

4.2.2. Основні завдання бізнес-аналітика

Бізнес-аналітик виконує низку ключових функцій, що забезпечують правильне формування вимог та успішну реалізацію IT-проєктів.

1. Збір і формалізація вимог:

✓ Збір вимог здійснюється через інтерв'ю, воркшопи, опитування, аналіз документів і бізнес-процесів.

✓ Формалізація означає перетворення неструктурованих ідей у чіткі, зрозумілі та перевірювані вимоги.

✓ Результатом є специфікації, user stories, use cases, діаграми UML чи BPMN.

Приклад: замовник каже «хочу систему, яка підвищить продажі», BA формалізує це у вимогу: «Система має забезпечувати автоматичну розсилку спеціальних пропозицій клієнтам залежно від історії їхніх покупок».

2. Комунікація зі стейкхолдерами:

✓ BA виступає каналом зв'язку між усіма зацікавленими сторонами.

✓ Він забезпечує, щоб усі учасники однаково розуміли вимоги, навіть якщо вони мислять різними «мовами» (бізнес – у категоріях прибутку, команда – у категоріях технологій).

✓ Важлива частина цієї роботи – фасилітація зустрічей, уміння задавати уточнюючі питання, уникати двозначностей.

3. Виявлення бізнес-цілей і трансформація їх у технічні рішення:

✓ Бізнес-аналітик аналізує справжні цілі організації, щоб уникнути ситуацій, коли команда вирішує «симптоми», а не «причину».

✓ BA разом із замовником формує бізнес-кейс і визначає, яку цінність має приносити продукт.

✓ Потім він трансформує ці цілі у вимоги до системи, які можуть бути реалізовані технічно.

Приклад: бізнес-ціль – «зменшити кількість помилок при обробці замовлень». Технічне рішення – «автоматизувати перевірку введених даних через інтеграцію з базою клієнтів».

4. Підтримка життєвого циклу вимог *рис.4.4):

✓ Вимоги постійно змінюються у процесі проєкту, і BA супроводжує їх на всіх етапах: від ініціації до релізу.

✓ Він відповідає за актуальність документації, а також за трасування вимог – можливість відстежити, як кожна бізнес-потреба була реалізована у продукті.

✓ У Agile-середовищі BA допомагає вести беклог і працює разом із Product Owner над уточненням user stories.

Рисунок 4.4 – Життєвий цикл вимог при супроводі BA

4.2.3. Ключові компетенції бізнес-аналітика

Робота бізнес-аналітика охоплює різні напрями – від комунікації з замовником до створення технічних моделей систем. Тому для успішної діяльності BA необхідно володіти як soft skills (м'якими навичками), так і hard skills (технічними навичками) (табл.4.1).

1. Soft skills (м'які навички)

Це навички, що визначають здатність ефективно взаємодіяти з людьми та організовувати комунікацію.

✓ Комунікація.

Бізнес-аналітик має бути здатним чітко і зрозуміло пояснювати складні ідеї, як замовнику, так і технічній команді.

Важливо не лише говорити, а й слухати: уміння задавати уточнюючі питання допомагає уникнути непорозумінь.

✓ Фасилітація.

BA часто проводить зустрічі, воркшопи чи брейнсторми, де потрібно спрямовувати дискусію, щоб команда та замовник прийшли до узгодженого рішення.

Тут важливі нейтральність, організаційні здібності та вміння працювати з груповою динамікою.

✓ Аналітичне та критичне мислення.

BA має бачити причинно-наслідкові зв'язки та виявляти корінь проблеми, а не лише симптоми.

✓ Емпатія та переговорні навички.

Здатність зрозуміти точку зору замовника чи розробника допомагає будувати довіру й зменшувати конфлікти.

✓ 2. Hard skills (технічні навички)

Це інструментарій, який дозволяє BA формалізувати вимоги, створювати специфікації та працювати разом із технічною командою.

✓ Моделювання бізнес-процесів (BPMN, UML).

За допомогою схем і діаграм BA відображає логіку бізнес-процесів, взаємодію користувачів із системою, сценарії роботи.

Це допомагає як замовникам, так і розробникам бачити єдину картину.

✓ SQL та робота з базами даних.

У багатьох проєктах BA має вміти формулювати прості запити до бази даних, щоб аналізувати дані, перевіряти гіпотези чи підготовлювати приклади для тестування.

✓ UML (Unified Modeling Language).

Використовується для побудови діаграм класів, варіантів використання, послідовностей взаємодій між компонентами системи.

✓ Знання інструментів.

Jira, Confluence, Trello, Figma, Enterprise Architect – це середовище, у якому BA документує вимоги, відстежує задачі та співпрацює з командою.

Таблиця 4.1 – Навички «Soft та Hard Skills»

Soft skills (м'які навички)	Hard skills (технічні навички)
Комунікація (уміння пояснювати і слухати)	Моделювання бізнес-процесів (BPMN, UML)
Фасилітація (проведення зустрічей, воркшопів)	SQL та робота з базами даних
Аналітичне та критичне мислення	UML-діаграми: use cases, класів, послідовності
Емпатія та переговорні навички	Інструменти: Jira, Confluence, Figma, EA

4.2.4. Приклади: роль BA у стартапі vs у великій корпорації

1. Бізнес-аналітик у стартапі

У невеликих командах BA часто поєднує кілька функцій одночасно: частково менеджера проєкту, UX-дизайнера та навіть тестувальника.

Особливості:

✓ Гнучкість. У стартапах вимоги змінюються швидко, тому BA має постійно уточнювати беклог та пріоритети.

✓ Мінімум формалізації. Замість великих документів BA створює короткі user stories, діаграми або прототипи.

✓ Швидкий зворотний зв'язок. BA безпосередньо комунікує з засновниками та кінцевими користувачами.

✓ Фокус на цінності. Завдання BA – допомогти команді швидко випустити MVP (мінімально життєздатний продукт).

Приклад: у стартапі, що створює мобільний застосунок для доставки їжі, BA збирає відгуки користувачів і трансформує їх у швидкі зміни інтерфейсу.

2. Бізнес-аналітик у великій корпорації

У великих компаніях BA зазвичай працює у великих проєктних командах, де є чіткі ролі та стандарти.

Особливості:

✓ Висока формалізація. Необхідні детальні специфікації, регламенти, погоджувальні процедури.

✓ Робота з багатьма стейкхолдерами. BA координує інтереси відділів (маркетинг, фінанси, IT, безпека).

✓ Трасування вимог. Вимоги документуються і відстежуються протягом усього життєвого циклу.

✓ Спеціалізація. BA може бути вузько спрямованим: працювати лише з вимогами до інтеграцій, даних або UX.

✓ Взаємодія з глобальною командою. У корпораціях BA часто комунікує з колегами в різних країнах.

У таблиці 4.2 наведено порівняння особливостей роботи BA у стартапі та великій корпорації.

Приклад: у банківській корпорації BA відповідає за вимоги до інтеграції мобільного застосунку з CRM-системою, веде документацію за стандартами ISO.

Таблиця 4.2. – Порівняння: Стартап та Корпорація

Ознака	Стартап	Корпорація
Формалізація	Мінімальна (user stories, прототипи)	Висока (специфікації, документи)
Гнучкість	Дуже висока, швидкі зміни	Обмежена регламентами
Стейкхолдери	Засновники, користувачі	Відділи, керівництво, регулятори
Фокус	MVP, швидкий результат	Якість, стабільність, відповідність стандартам
Роль BA	Універсал (частково PM/UX)	Спеціалізований аналітик

4.3. Інструменти збору вимог

Для формування вимог до проєкту використовуються різноманітні методи.

4.3.1. Інтерв'ю та опитування: як готувати питання, приклади відкритих і закритих питань

1. Інтерв'ю як метод збору вимог

Інтерв'ю – це особиста розмова бізнес-аналітика зі стейкхолдером або користувачем, мета якої – з'ясувати потреби, очікування та проблеми.

✓ Формати: індивідуальні (one-to-one), групові (focus group).

✓ Коли застосовують: на ранніх етапах проєкту, коли необхідно зрозуміти бізнес-контекст і «біль» замовника.

✓ Переваги: можливість уточнити відповіді, задати додаткові питання, зібрати глибинну інформацію.

✓ Недоліки: трудомісткість, залежність від відкритості співрозмовника.

Як готуватися до інтерв'ю:

✓ Визначити цілі розмови: які саме питання потрібно з'ясувати.

✓ Скласти список основних тем (не зачитувати питання механічно).

✓ Вибрати формат: структуроване (чіткі питання), напівструктуроване, вільне.

✓ Створити комфортні умови для співрозмовника.

2. Опитування (анкети)

Опитування – це письмовий збір інформації за допомогою анкет чи онлайн-форм (Google Forms, Typeform, Microsoft Forms).

✓ Коли застосовують: якщо треба швидко зібрати дані від великої кількості користувачів.

✓ Переваги: масштабованість, швидкість, можливість статистичного аналізу.

✓ Недоліки: відповіді можуть бути поверхневими, немає можливості уточнювати деталі.

Як готувати опитування:

✓ Формулювати питання простою мовою, уникати професійного жаргону.

✓ Не змішувати кілька питань в одне.

✓ Використовувати логічну структуру (від загального до конкретного).

✓ Перед запуском – протестувати анкету на кількох людях.

Типи питань (табл.4.3):

1. Відкриті питання (open-ended)

✓ Дають можливість респонденту відповісти у вільній формі.

✓ Використовуються для виявлення нових ідей і проблем.

Приклади:

✓ «Які труднощі ви відчуваєте під час роботи із CRM-системою?»

✓ «Що б ви хотіли змінити в процесі реєстрації клієнтів?»

2. Закриті питання (closed-ended)

✓ Передбачають обмежений набір відповідей.

✓ Використовуються для кількісного аналізу та статистики.

Приклади:

✓ «Як часто ви користуєтеся мобільним застосунком?»

✓ щодня / кілька разів на тиждень / рідше

✓ «Чи задоволені ви швидкістю роботи системи?»

✓ так / ні / частково

У таблиці 4.3 наведені приклади відкритих та закритих запитань прі проведенні опитування.

Таблиця 4.3 – Відкриті та закриті питання із прикладами.

Відкриті питання	Закриті питання
Які труднощі ви відчуваєте під час роботи із CRM-системою?	Як часто ви користуєтеся мобільним застосунком? (щодня / кілька разів на тиждень / рідше)
Що б ви хотіли змінити в процесі реєстрації клієнтів?	Чи задоволені ви швидкістю роботи системи? (так / ні / частково)

4.3.2. Воркшопи та мозкові штурми: фасилітація групових сесій

1. Що таке воркшоп у бізнес-аналізі

Воркшоп (workshop) – це структурована робоча зустріч, у якій беруть участь бізнес-замовники, кінцеві користувачі, розробники, тестувальники та інші стейкхолдери.

Його мета – спільно напрацювати рішення, визначити вимоги або узгодити бачення продукту.

Особливості воркшопів:

- ✓ Участь кількох зацікавлених сторін одночасно.
- ✓ Висока інтерактивність (дискусії, вправи, прототипування).
- ✓ Конкретний результат: перелік вимог, прототип, пріоритизований беклог.

2. озковий штурм як метод збору ідей

Мозковий штурм (brainstorming) – це групова техніка генерації ідей.

Принцип: спочатку кількість, потім якість. Спочатку збираються всі ідеї, навіть «фантастичні», а потім відбираються реалістичні.

Етапи мозкового штурму:

- ✓ Формулювання проблеми або питання.
- ✓ Генерація ідей (без критики).
- ✓ Сортування й кластеризація.
- ✓ Вибір найперспективніших.

Використовується для: пошуку нових функцій продукту, визначення пріоритетів, створення сценаріїв користувачів.

3. Роль фасилітатора (бізнес-аналітика)

Фасилітатор (часто це BA) відповідає за організацію та ефективність

групової сесії (рис.4.5):

- ✓ ставить чіткі цілі зустрічі;
- ✓ слідкує за регламентом і темпом роботи;
- ✓ забезпечує рівну участь усіх учасників;
- ✓ нейтрально керує дискусією, не просуваючи власних ідей;
- ✓ фіксує результати у вигляді вимог, схем, беклогу.

Переваги групових методів

- ✓ Залучають різні точки зору (бізнес, технічна команда, користувачі).
- ✓ Дозволяють швидко отримати великий обсяг інформації.
- ✓ Допомагають досягнути узгодженого бачення.

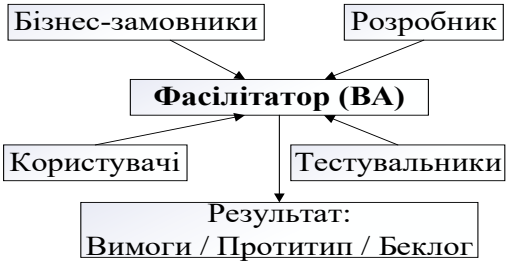

Рисунок 4.5 – Фасилітація групової сесії

4.3.3. Прототипи та wireframes

Wireframe – це «каркас» інтерфейсу, спрощена схема екрана без графічних деталей. Вона показує структуру сторінки, розташування кнопок, полів, меню.

Прототип – це більш деталізована й інтерактивна модель майбутнього продукту, яка може імітувати роботу системи.

Обидва інструменти допомагають замовникам і команді бачити майбутній продукт ще до його розробки (рис.4.6).

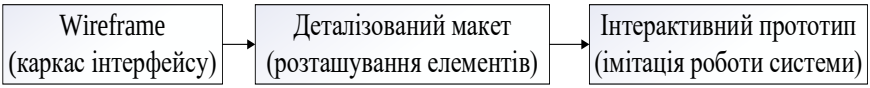

Рисунок 4.6 – Від wireframe до прототипу

Навіщо вони потрібні

✓ Допомагають уточнити вимоги ще до написання коду.

✓ Знижують ризик непорозумінь між бізнесом і командою.

✓ Дають змогу користувачам протестувати сценарії ще на ранньому етапі.

✓ Є дешевим способом перевірки ідей.

Інструменти створення прототипів:

✓ Figma – популярний інструмент для створення інтерактивних прототипів у браузері.

✓ Balsamiq – спеціалізований софт для швидкого створення wireframes у стилі «скетчів».

✓ Adobe XD, Sketch – професійні інструменти з багатими можливостями для дизайну.

Приклади застосування:

1. У проєкті CRM-системи wireframes допомогли узгодити з клієнтом розташування полів і кнопок до початку розробки.

2. У стартапі з онлайн-освіти інтерактивний прототип у Figma показали інвесторам, щоб отримати фінансування.

4.3.4. Юзер-сторі

Юзер-сторі (user stories) – це короткі описи вимог до системи з точки зору кінцевого користувача.

Вони відповідають на три ключові питання (рис.4.7):

✓ Хто користувач?

✓ Що він хоче зробити?

✓ Навіщо йому це потрібно?

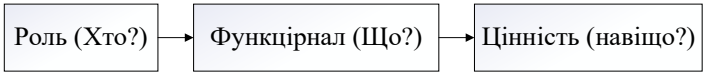

Рисунок 4.7 – «Структура юзер-сторі»:

Формат класичної юзер-сторі:

«As a [роль], I want [функціонал], so that [бізнес-цінність]».

Або українською:

«Як [роль], я хочу [дія/функціонал], щоб [результат/цінність]».

Чому юзер-сторі важливі

✓ Орієнтують команду на цінність для користувача, а не лише на технічні деталі.

✓ Легко зрозумілі для всіх стейкхолдерів.

✓ Гнучкі: їх можна швидко змінювати й доповнювати.

✓ Є базою для побудови беклогу продукту в Scrum чи Kanban.

Приклади юзер-сторі

1. Як покупець, я хочу зберегти товари у «Вибране», щоб купити їх пізніше.

2. Як студент, я хочу переглядати свої оцінки онлайн, щоб бачити прогрес у навчанні.

3. Як адміністратор системи, я хочу мати звіт про активність користувачів, щоб виявляти підозрілі дії.

Критерії прийнятності (Acceptance Criteria)

Кожна юзер-сторі доповнюється умовами, які пояснюють, що саме вважається виконаним.

Приклад:

✓ Юзер-сторі: «Як користувач, я хочу відновити пароль через email, щоб мати доступ до системи».

✓ Acceptance Criteria:

1. Користувач може ввести email на формі.
2. Система надсилає лист із посиланням для відновлення.
3. Після переходу за посиланням користувач задає новий пароль.

4.3.5. Use cases: опис сценаріїв використання

Use case (сценарій використання) – це опис взаємодії між користувачем (актором) і системою для досягнення конкретної мети.

Він показує, як саме система використовується у реальних ситуаціях.

Структура Use case

Класичний опис сценарію використання містить:

1. Назву (що описує ціль, наприклад: «Оформлення замовлення»).

2. Акторів (користувач, адміністратор, система).
3. Передумови (що має бути виконано перед початком).
4. Основний сценарій (послідовність кроків взаємодії).
5. Альтернативні сценарії (що робити, якщо щось пішло не так).
6. Результат (очікуваний вихід після виконання).

Приклад Use case

Назва: Авторизація користувача.

✓ Актор: Користувач.

✓ Передумови: Користувач має зареєстрований акаунт.

✓ Основний сценарій:

1. Користувач відкриває сторінку входу.
2. Вводить логін і пароль.
3. Система перевіряє дані.
4. Якщо дані правильні – користувач потрапляє до кабінету.

✓ Альтернативний сценарій: Якщо дані неправильні – система виводить повідомлення про помилку.

✓ Результат: Користувач успішно авторизований.

Діаграма Use case (UML)

Use cases часто зображаються у вигляді UML-діаграм (рис.4.8):

✓ Актори позначаються «чоловічками».

✓ Сценарії – еліпсами.

✓ Лінії показують взаємодію актора із системою.

Наприклад:

✓ Актор «Клієнт» взаємодіє зі сценаріями «Зареєструватися», «Авторизуватися», «Замовити товар».

✓ Актор «Адміністратор» має сценарій «Керувати замовленнями».

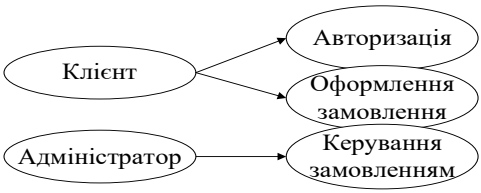

Рисунок 4.8 – UML Use case діаграми

Коли використовують Use cases

✓ Для опису функціональних вимог.

✓ Для уточнення взаємодії в складних системах.

✓ Для узгодження між бізнесом і технічною командою.

4.3.6. Порівняння підходів: коли який інструмент кращий

1. Інтерв'ю та опитування

✓ Коли кращі: на початкових етапах, коли потрібно зрозуміти загальне бачення, проблеми й очікування.

✓ Сильні сторони: глибоке занурення у проблематику, можливість уточнень.

✓ Обмеження: часозатратні, залежать від якості питань і готовності співрозмовника.

2. Воркшопи та мозкові штурми

✓ Коли кращі: коли потрібно швидко узгодити вимоги між кількома стейкхолдерами або створити спільне бачення продукту.

✓ Сильні сторони: залучення різних точок зору, колективна генерація ідей.

✓ Обмеження: потребують фасилітації та чіткого регламенту, ризик «домінування» одних учасників.

3. Прототипи та wireframes

✓ Коли кращі: коли є ризик різного розуміння вимог бізнесом і командою; для перевірки UX.

✓ Сильні сторони: візуалізація, можливість тестування інтерфейсу ще до розробки.

✓ Обмеження: потребують додаткового часу й навичок дизайну.

4. Юзер-сторі

✓ Коли кращі: у гнучких методологіях (Scrum, Kanban), де важлива швидкість і адаптивність.

✓ Сильні сторони: простота, орієнтація на цінність користувача, зрозумілі для всіх.

✓ Обмеження: можуть бути надто загальними без чітких критеріїв прийнятності.

5. Use cases

✓ Коли кращі: у складних системах з багатьма сценаріями, інтеграціями та акторами.

✓ Сильні сторони: формалізований опис взаємодії, підходять для тестування.

✓ Обмеження: можуть бути громіздкими, займають час на деталізацію.

Таблиця 4.4 – Порівняння інструментів

Інструмент	Коли застосовувати	Переваги	Обмеження
Інтерв'ю та опитування	На старті проєкту, для виявлення потреб	Глибокі відповіді, контакт зі стейкхолдером	Часозатратні, залежать від відкритості
Воркшопи, мозкові штурми	Для узгодження вимог у групі	Колективна генерація ідей	Потребують фасилітації
Прототипи, wireframes	Для перевірки UX, візуалізації	Зменшують ризик непорозумінь	Потребують часу і навичок
Юзер-сторі	Agile-проекти, швидкі зміни	Простота, гнучкість	Можуть бути нечіткими
Use cases	Складні системи	Формалізовані сценарії	Громіздкі, деталізація займає час

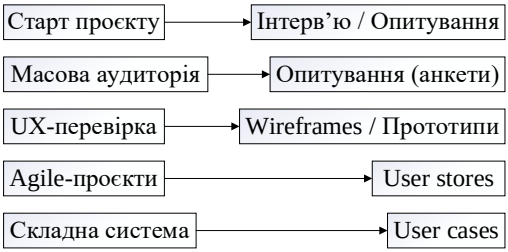

Рисунок 4.9 – Вибір інструмента залежно від контексту

4.4. Трасування вимог

Трасування вимог (requirements traceability) – це процес встановлення та підтримання зв'язків між вимогами і всіма артефактами проєкту: бізнес-цілями, функціональними специфікаціями, тестами, кодом, випусками продукту.

Простіше кажучи, трасування відповідає на запитання (рис.4.10):

- ✓ Звідки взялася ця вимога?
- ✓ Яку бізнес-мету вона реалізує?
- ✓ Де і як вона реалізована у системі?
- ✓ Чи протестована вона?

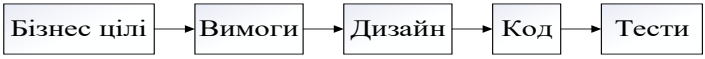

Рисунок 4.10 – Ланцюг трасування вимог»:

Навіщо потрібне трасування:

1. Прозорість і контроль. Замовник і команда завжди знають, які вимоги реалізовані, а які ще в роботі.

2. Управління змінами. Якщо змінюється вимога, ВА може швидко оцінити, на які частини системи це вплине.

3. Якість тестування. Тест-кейси будуються на основі вимог, а трасування допомагає переконатися, що всі вимоги протестовані.

4. Відповідність стандартам. У багатьох галузях (медицина, авіація, фінанси) аудитори вимагають доказів, що кожна бізнес-вимога простежується до реалізації.

5. Мінімізація ризиків. Трасування допомагає уникати «забутих» або «зайвих» вимог.

Типи трасування:

- ✓ Пряме трасування (forward traceability). Від бізнес-цілі → до вимоги → до дизайну → до коду → до тестів.
- ✓ Зворотне трасування (backward traceability). Від тесту → назад до вимоги → до бізнес-цілі (чи справді ця функція потрібна?).
- ✓ Двонаправлене трасування (bidirectional). Поєднує обидва підходи й

дає повну картину.

Приклад. Бізнес-вимога: «Покращити безпеку доступу користувачів».

✓ Функціональна вимога: «Система має підтримувати двофакторну автентифікацію»:

✓ Дизайн: форма для введення коду.

✓ Код: модуль перевірки OTP.

✓ Тест-кейс: «При вході користувач вводить код із SMS».

Завдяки трасуванню можна простежити шлях від бізнес-цілі до конкретного тесту.

4.4.1. Інструменти трасування вимог

1. Матриця трасування вимог (RTM – Requirements Traceability Matrix)

Матриця трасування – це таблиця, яка пов'язує вимоги з іншими артефактами: тестами, кодом, бізнес-цілями, яка містить наступні ID вимоги (табл.4.5):

✓ Опис вимоги

✓ Джерело (зв'язок із бізнес-ціллю)

✓ Стан реалізації

✓ Пов'язані тест-кейси

Переваги: проста у створенні, зрозуміла всім учасникам.

Недоліки: складно підтримувати в актуальному стані при великих проєктах.

Таблиця 4.5 – Приклад RTM (спрощений)

ID	Вимога	Бізнес-ціль	Реалізація	Тест-кейс
RQ-01	Двофакторна автентифікація	Підвищення безпеки	Виконано	TC-15
RQ-02	Збереження історії транзакцій	Контроль операцій	У розробці	TC-22, TC-23

2. Інструменти керування вимогами

Для автоматизації трасування використовують спеціалізовані системи:

✓ Jira + плагіни (Xray, Zephyr). Дозволяють пов'язувати вимоги, задачі

та тести.

✓ IBM DOORS. Потужна система для керування вимогами у великих корпоративних проєктах.

✓ Helix RM, Polarion. Підтримують двонаправлене трасування та роботу з великими наборами вимог.

✓ Confluence. Використовується як репозиторій вимог з можливістю посилань на Jira та інші інструменти.

3. Автоматизація трасування

✓ В сучасних системах кожна вимога має унікальний ID.

✓ Цей ID «прикріплюється» до задачі в Jira, коду в Git, тесту в TestRail.

✓ Це дає змогу швидко простежити: якщо змінюється вимога → бачимо, які частини системи потрібно переробити.

4.4.2. Зв'язки між бізнес-вимогою → функціональною вимогою → технічною специфікацією → тест-кейсами

1. Бізнес-вимоги

✓ Формуються на рівні організації та відображають стратегічні цілі бізнесу.

✓ Відповідають на запитання: «Навіщо ми реалізуємо цей проєкт?»

✓ Приклад: «Підвищити безпеку доступу клієнтів до онлайн-банкінгу».

2. Функціональні вимоги

✓ Випливають із бізнес-вимог і описують що система має робити, щоб задовольнити бізнес-цілі.

✓ Формулюються у вигляді функцій або сценаріїв.

Приклад: «Система має підтримувати двофакторну автентифікацію при вході».

3. Технічні специфікації

✓ Це деталізовані описи того, як саме реалізується функціональна вимога на рівні технологій, архітектури та дизайну.

✓ Містять модулі, API, алгоритми, інтеграції.

Приклад: «Розробити модуль OTP, який генерує код тривалістю 60 секунд і надсилає його через SMS-шлюз».

4. Тест-кейси

✓ Це сценарії перевірки, які дозволяють підтвердити, що вимога реалізована коректно.

✓ Кожен тест-кейс прив'язується до конкретної функціональної вимоги.

Приклад:

✓ ТС-01: Перевірити, що після введення правильного OTP користувач авторизується.

✓ ТС-02: Перевірити, що після введення неправильного OTP система блокує доступ.

Зв'язок у вигляді ланцюга (рис.4.11):

1. Бізнес-вимога задає «Навіщо».
2. Функціональна вимога описує «Що».
3. Технічна специфікація пояснює «Як».
4. Тест-кейси перевіряють «Чи працює це правильно».

Таким чином, кожна вимога має повний «слід» – від бізнес-цілі до конкретного тесту, що гарантує відповідність результату очікуванням.

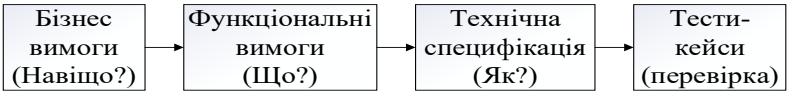

Рисунок 4.11 – Ланцюг Вимога → Реалізація → Перевірка

4.4.3. Використання матриці трасування вимог (RTM – Requirements Traceability Matrix)

Матриця трасування вимог (Requirements Traceability Matrix, RTM) – це таблиця, яка показує зв'язки між вимогами, тестами, бізнес-цілями та іншими артефактами проєкту.

Її основна мета – гарантувати, що кожна вимога реалізована та протестована, а також що жодна зайва функція не розроблена.

Для чого використовується RTM

1. Контроль виконання. Дозволяє відстежувати статус кожної вимоги.
2. Повнота тестування. Перевіряє, чи всі вимоги мають відповідні

тест-кейси.

3. Управління змінами. Якщо вимога змінюється, матриця показує, які інші артефакти потрібно змінити.

4. Прозорість для замовника. Легко демонструвати прогрес і покриття вимог на рев'ю чи аудиті.

Структура RTM (табл.4.6, рис.4.12). Класична RTM містить такі колонки:

- ✓ ID вимоги
- ✓ Опис вимоги
- ✓ Тип вимоги (бізнес / функціональна / нефункціональна)
- ✓ Джерело (посилання на бізнес-ціль чи документ)
- ✓ Статус реалізації (виконано, у роботі, відкладено)
- ✓ Відповідні тест-кейси
- ✓ Коментарі

Таблиця 4.6 – Приклад RTM (спрощений)

ID	Вимога	Тип	Джерело	Статус	Тест-кейси
RQ-01	Двофакторна автентифікація	Функціональна	Бізнес-ціль: Безпека	Виконано	TC-01, TC-02
RQ-02	Історія транзакцій	Функціональна	Бізнес-ціль: Контроль	У розробці	TC-10, TC-11
RQ-03	Час відгуку < 2 секунд	Нефункціональна	SLA-договір	Планується	TC-20

Рисунок 4.12 – RTM як центр зв'язків», де показано:

Особливості ведення RTM:

- ✓ У невеликих проєктах RTM можна вести у Excel або Google Sheets.

✓ У великих – інтегрувати в Jira, IBM DOORS, Polarion, TestRail.

✓ Оновлювати RTM потрібно регулярно, щоб вона відображала актуальний стан проєкту.

4.4.4. Інструменти: Jira, Confluence, IBM DOORS

1. Jira

Призначення: система управління завданнями та Agile-проєктами.

Можливості у трасуванні вимог:

✓ Створення епіків, юзер-сторі, підзадач.

✓ Зв'язки між завданнями (linking): «relates to», «blocks», «is blocked by».

✓ Плагіни (Xray, Zephyr) дозволяють пов'язувати вимоги із тест-кейсами.

✓ Звіти та діаграми прогресу по реалізації вимог.

-Особливість: зручно використовувати в Agile-командах для двонаправленого трасування (від вимоги до тесту і назад).

2. Confluence

Призначення: корпоративний Wiki-простір від Atlassian.

Можливості у трасуванні вимог:

✓ Зберігання бізнес-вимог, специфікацій та документів.

✓ Інтеграція з Jira: сторінки Confluence можна пов'язати з конкретними задачами у Jira.

✓ Підтримка шаблонів (наприклад, для вимог або RTM).

Особливість: виступає як «репозиторій вимог» із можливістю гнучкого документування й колективної роботи.

3. IBM DOORS

Призначення: спеціалізована система для управління вимогами у великих і критично важливих проєктах (авіація, автомобілебудування, оборона, медицина).

Можливості у трасуванні вимог:

✓ Повна підтримка двонаправленого трасування.

✓ Робота з тисячами і навіть мільйонами вимог.

✓ Можливість імпорту/експорту з інших систем (Excel, Word).

✓ Контроль версій і аудит змін.

Особливість: використовується там, де потрібна сертифікація, відповідність стандартам (ISO, CMMI, FDA) і максимальна прозорість.

Порівняння інструментів трасування наведено у таблиці 4.7, а на рисунку 4.12 зображена їх взаємодія.

Таблиця 4.7 – Порівняння інструментів

Інструмент	Для яких проєктів підходить	Переваги	Обмеження
Jira	Agile-команди, середні проєкти	Інтеграція з тестами, прозорість, гнучкість	Потребує плагінів для повного трасування
Confluence	Документування, інтеграція з Jira	Зручне зберігання вимог, Wiki-простір	Сама по собі не виконує трасування
IBM DOORS	Великі, регульовані галузі	Потужність, аудит, сертифікація	Складність у використанні, висока вартість

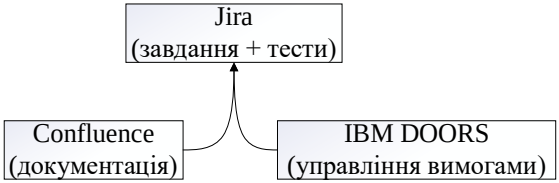

Рисунок 4.13 – Роль інструментів у трасуванні

4.4.5. Приклад RTM у простому проєкті (CRM-система)

Опис ситуації.

Уявімо розробку базової CRM-системи для невеликої компанії, яка працює з клієнтами.

Замовник ставить перед бізнес-аналітиком три основні цілі:

1. Забезпечити збереження даних про клієнтів.
2. Реалізувати управління угодами та контактами.
3. Дати можливість аналітики продажів.

Бізнес-вимоги → Функціональні вимоги → Тест-кейси

BR-01. Збереження інформації про клієнтів.

✓ FR-01: Система дозволяє створювати, редагувати та видаляти картку клієнта.

✓ Тест-кейси: TC-01 (створення картки), TC-02 (редагування), TC-03 (видалення).

BR-02. Управління угодами.

✓ FR-02: Система дозволяє створювати угоду та прив'язувати її до клієнта.

✓ Тест-кейси: TC-04 (створення угоди), TC-05 (зв'язок угоди з клієнтом).

BR-03. Аналітика продажів.

✓ FR-03: Система формує звіт про кількість угод за період.

✓ Тест-кейси: TC-06 (перевірка звіту за місяць), TC-07 (перевірка звіту за рік).

Таблиця 4.8 – Приклад RTM для CRM

ID	Бізнес-вимога	Функціональна вимога	Статус	Тест-кейси
BR-01	Збереження даних про клієнтів	FR-01: CRUD для клієнтів	Виконано	TC-01, TC-02, TC-03
BR-02	Управління угодами	FR-02: Створення угод	У розробці	TC-04, TC-05
BR-03	Аналітика продажів	FR-03: Формування звітів	Планується	TC-06, TC-07

Висновок. Матриця трасування навіть у простому проєкті допомагає:

✓ Відстежити, чи всі бізнес-вимоги мають функціональні реалізації.

✓ Побачити, чи кожна функціональна вимога покрита тестами.

✓ Контролювати прогрес реалізації (виконано / у роботі / планується).

4.4.6. Приклади трасування у реальних IT-проєктах

1. Фінансові системи (банківський софт)

Контекст: висока регульованість, відповідність стандартам (ISO 27001, PCI DSS).

Трасування:

✓ Бізнес-вимога: «Забезпечити безпечні транзакції».

✓ Функціональна вимога: «Верифікація операцій через OTP».

✓ Технічна специфікація: модуль SMS-шлюзу.

✓ Тест-кейси: «Перевірити коректність відправки OTP».

Особливість: аудитори перевіряють наявність повного ланцюга трасування.

2. Авіаційна промисловість

Контекст: критичні системи, сертифікація FAA/EASA, стандарти DO-178C.

Трасування:

✓ Бізнес-вимога: «Підвищити безпеку польотів».

✓ Функціональна вимога: «Система має контролювати висоту польоту у реальному часі».

✓ Технічна специфікація: модуль сенсорів висоти.

✓ Тест-кейси: «Перевірити реакцію на падіння висоти >100 м за 5 секунд».

Особливість: повна двонаправлена трасованість є обов’язковою умовою сертифікації.

3. Медичні системи

Контекст: відповідність стандартам HIPAA, GDPR, FDA.

Трасування:

✓ Бізнес-вимога: «Захист персональних медичних даних».

✓ Функціональна вимога: «Шифрування бази даних пацієнтів».

✓ Технічна специфікація: алгоритм AES-256.

✓ Тест-кейси: «Перевірити шифрування/дешифрування записів».

Особливість: у медичних проєктах трасування використовується для доведення відповідності регуляторним вимогам.

4. E-commerce платформа

Контекст: велика кількість користувачів, швидке оновлення фіч.

Трасування:

✓ Бізнес-вимога: «Покращити конверсію продажів».

✓ Функціональна вимога: «Реалізувати функцію рекомендацій товарів».

✓ Технічна специфікація: інтеграція ML-моделі.

✓ Тест-кейси: «Перевірити відображення 5 рекомендованих товарів у кошику».

Особливість: у таких проєктах трасування допомагає балансувати швидкість розробки й контроль вимог.

4.5. Управління змінами у вимогах

Чому вимоги завжди змінюються?

1. Динамічне бізнес-середовище

✓ Бізнес-процеси компаній постійно еволюціонують.

✓ Поява нових конкурентів, зміни на ринку, впровадження нових технологій – усе це змінює пріоритети замовника.

Приклад: стартап запускає мобільний застосунок, але конкуренти додають нову можливість – замовник змінює вимоги, щоб не відставати.

2. Неповнота початкових вимог

✓ Неможливо одразу зібрати усі вимоги. Частина з них виявляється вже під час розробки.

✓ Замовники можуть не до кінця розуміти свої потреби на старті.

Приклад: у CRM спочатку вимагали «зберігати клієнтів», але пізніше з’явилась потреба у «сегментації клієнтів».

3. Нові технології та обмеження

✓ У процесі роботи можуть з’являтися нові інструменти або зміни в технічних обмеженнях.

Приклад: вибір бази даних. Спочатку – MySQL, але при масштабуванні довелось перейти на PostgreSQL.

4. Регуляторні вимоги

✓ Закони та стандарти часто змінюються.

✓ У проєктах з фінансів, медицини, e-government доводиться

підлаштовувати систему під нові юридичні норми.

Приклад: зміни у GDPR змушують додати функцію «видалити всі дані користувача».

5. Людський фактор

✓ Замовники можуть змінювати своє бачення.

✓ Змінюється команда: новий Product Owner чи керівник може мати інші пріоритети.

Приклад: новий менеджер вирішує, що мобільна версія є важливішою за веб.

6. Зворотний зв'язок від користувачів

✓ Після запуску MVP або бета-тестування користувачі дають реальні відгуки.

✓ Часто це призводить до перегляду функціоналу.

Приклад: користувачі незадоволені складною формою реєстрації → BA ініціює зміни у вимогах.

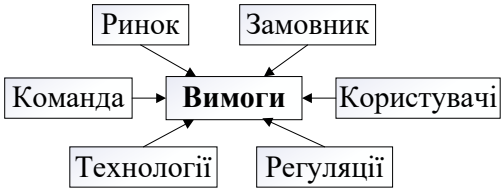

Рисунок 4.14 – Джерела змін вимог

4.5.1. Процес управління змінами (Change Request)

Change Request (CR) – це офіційний запит на зміну вимоги, функціоналу чи інших характеристик проєкту. Він ініціюється, коли з’являється нова потреба, помилка у вимогах або необхідність адаптації під зовнішні умови.

Життєвий цикл Change Request (рис.4.15):

1. Ініціація

✓ Хтось зі стейкхолдерів (замовник, Product Owner, тестувальник, розробник) формує запит.

✓ CR описується у спеціальній формі або інструменті (Jira, Confluence, ServiceNow).

2. Оцінка

✓ Бізнес-аналітик аналізує зміни: чи відповідають вони цілям проєкту, які артефакти зачіпають.

✓ Команда оцінює вартість і строки.

3. Прийняття рішення

✓ Change Control Board (CCB) або керівник проєкту ухвалює:

✓ ☑ прийняти;

✓ ❌ відхилити;

✓ ⏳ відкласти.

4. Реалізація

✓ Внесення змін у вимоги, документацію, код.

✓ Оновлення RTM, щоб зафіксувати зв'язки.

5. Перевірка та закриття

✓ Виконуються тест-кейси, які підтверджують реалізацію.

✓ CR закривається в системі.

Рисунок 4.15 – Запит Request Lifecycle:

Типова структура Change Request

✓ ID зміни.

✓ Автор (хто ініціював).

✓ Опис зміни.

✓ Причина (чому потрібна зміна).

✓ Вплив на бізнес та технічні характеристики.

✓ Оцінка (вартість, час, ресурси).

✓ Статус (новий, у роботі, виконано, відхиллено).

4.5.2. Баланс між гнучкістю та стабільністю

1. Проблема протиріччя

В управлінні вимогами завжди існує конфлікт:

✓ З одного боку, гнучкість потрібна для швидкої реакції на зміни

бізнесу, ринку та користувачів.

✓ З іншого боку, надмірна кількість змін може дестабілізувати проєкт, призвести до хаосу в документації та зриву строків.

2. Гнучкість

✓ Дозволяє швидко адаптувати продукт під нові умови.

✓ Перевага в інноваційних галузях, стартапах, де пріоритет – швидкість.

Приклад: стартап на ранній стадії MVP вносить зміни щотижня на основі фідбеку користувачів.

3. Стабільність

✓ Забезпечує передбачуваність і контроль.

✓ Важлива у великих і регульованих проєктах (банківські, медичні, державні системи).

Приклад: у банківській системі не можна постійно змінювати вимоги, бо це загрожує безпеці та відповідності стандартам.

4. Як досягти балансу (рис.4.16)

1. Визначати «заморожені» вимоги – ті, що не можна змінювати без серйозного обґрунтування.

2. Використовувати пріоритизацію (MoSCoW, Kano) – щоб виділяти критичні та другорядні вимоги.

3. Встановлювати контроль змін через Change Request – зміни можливі, але проходять офіційний процес.

4. Гнучкість у дрібних фічах, стабільність у критичних функціях.

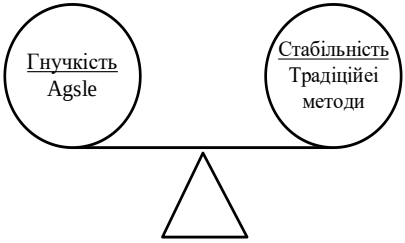

Рисунок 4.16 – Баланс гнучкість ↔ стабільність

Висновок. Правильний баланс між гнучкістю та стабільністю дозволяє

одночасно:

- ✓ зберегти адаптивність продукту,
- ✓ гарантувати якість, безпеку та передбачуваність.

4.5.3. Agile-підхід до змін (беклог як «живий документ»)

1. Зміни як норма в Agile

В Agile-філософії вважається, що зміни у вимогах – це природна частина процесу. Agile Manifesto прямо говорить: «Ми цінуємо реагування на зміни більше, ніж дотримання плану». Це означає, що замовник і команда мають бути готові до регулярного перегляду вимог.

2. Product Backlog – «живий документ»

У Scrum та інших Agile-методологіях усі вимоги збираються в беклог продукту. Беклог – це не статичний список, а постійно змінюваний документ, який:

- ✓ відображає поточні бізнес-пріоритети;
- ✓ оновлюється після кожного спринту чи отримання нового фідбеку;
- ✓ містить як нові вимоги, так і змінені або видалені.

Приклад: якщо користувачі просять спростити реєстрацію, BA змінює відповідний user story у беклозі, а Scrum-команда планує його в наступний спринт.

3. Пріоритизація змін

Зміни у беклозі пріоритизуються за допомогою технік:

- ✓ MoSCoW (Must, Should, Could, Won’t).
- ✓ Value vs Effort (матриця цінності та витрат).
- ✓ Kano model (мінімальні, очікувані та захопливі функції).

Це дозволяє відсіяти неважливі зміни й зосередитись на тих, що дають найбільшу цінність.

4. Роль BA в Agile-змінах

Бізнес-аналітик:

- ✓ збирає нові вимоги та фідбек,
- ✓ оновлює й уточнює user stories,
- ✓ допомагає Product Owner-у у пріоритизації,
- ✓ стежить, щоб нові вимоги не суперечили загальній цілі продукту.

Висновок. Agile-підхід до змін у вимогах базується на постійній еволюції беклогу (рис.4.17). Це дозволяє створювати продукт, який максимально відповідає реальним потребам користувачів і бізнесу.

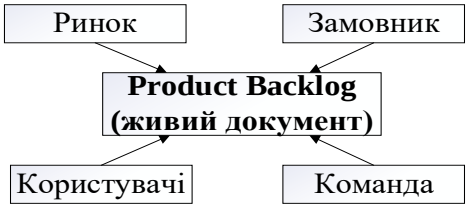

Рисунок 4.17 – Backlog – це постійно змінюючийся документ

4.5.4. Вплив змін на вартість і строки проєкту (закон Боема про зростання вартості змін із часом)

1. Суть закону Боема

Баррі Боем (Barry Boehm), один із піонерів інженерії програмного забезпечення, сформулював принцип: чим пізніше в життєвому циклі вноситься зміна у вимогу, тим дорожче вона коштує. Це пов'язано з тим, що на ранніх етапах (аналіз, проєктування) зміни можна внести відносно дешево. А на пізніх етапах (тестування, підтримка) будь-яка зміна тягне за собою переробку коду, документації, тестів і навіть архітектури.

2. Вплив на вартість

✓ На етапі аналізу: зміна вимоги може означати лише зміну тексту в документі (мінімальні витрати).

✓ На етапі розробки: доводиться переробляти код, інтеграції, базу даних.

✓ На етапі тестування: крім коду треба переробляти тест-кейси, автоматизацію, документацію.

✓ На етапі підтримки: зміни впливають на користувачів, потрібні міграції даних і додаткові витрати.

Приклад:

✓ На етапі аналізу зміна поля «Телефон» у CRM на «Телефон/Email» – кілька хвилин роботи.

✓ На етапі продакшену та інтеграцій – це може коштувати тисячі доларів і тижні часу.

3. Вплив на строки

Зміни на пізніх етапах можуть затримати реліз на тижні або місяці. Тому критично важливо фіксувати ключові вимоги на старті та контролювати їх зміни.

У Agile цей ризик зменшується за рахунок ітеративності: зміни вносяться поступово і тестуються одразу.

4. Візуалізація

Закон Боема зазвичай ілюструється графіком (рис.4.18).

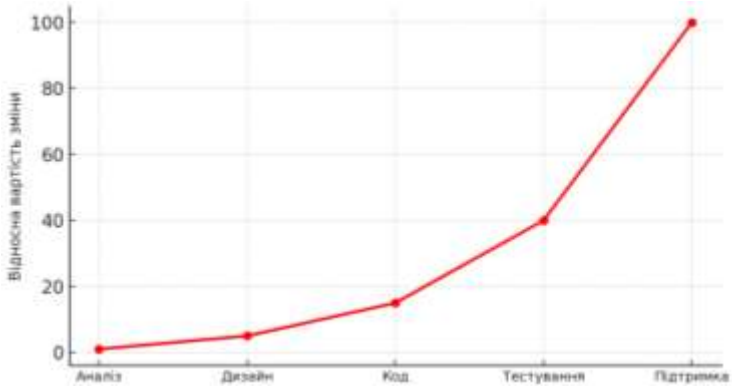

Рисунок 4.18 – графік «Закон Боема: зростання вартості змін із часом».

X-вісь: етапи життєвого циклу (Аналіз → Дизайн → Код → Тестування → Підтримка). Y-вісь: відносна вартість зміни. Сама крива різко зростає від етапу аналізу до підтримки.

Висновок

✓ Чим раніше виявлено і враховано зміни, тим дешевше і швидше їх реалізувати.

✓ BA повинен мінімізувати ризики пізніх змін через якісний збір вимог і управління беклогом.

4.5.5. Кращі практики управління змінами

До кращих практик управління змінами входять три основні складові (рис.4.19):

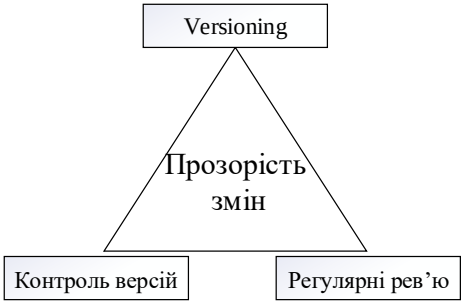

Рисунок 4.19 – Кращі практики управління змінами у вигляді трикутника:

1. Versioning (версіонування вимог)

Кожна зміна у вимогах має свою версію для чото використовується принцип «історії змін» (changelog).

Приклад:

- ✓ Документ SRS v1.0 – початкова версія.
- ✓ SRS v1.1 – додано нову вимогу до авторизації.
- ✓ SRS v2.0 – повністю переглянутий модуль безпеки.

Це дозволяє відслідковувати, коли і чому була внесена зміна, і хто її ініціював.

2. Контроль версій у документації

Для цього застосовуються інструменти:

- ✓ Confluence (зберігає історію редагувань).
- ✓ Git / SVN (для технічних специфікацій та артефактів).
- ✓ Jira (зв'язок завдань із вимогами).

Це гарантує, що команда працює з актуальною версією вимог, а не зі старою.

3. Регулярні рев'ю з клієнтом

Зміни повинні узгоджуватися із замовником. Регулярні сесії (щотижня або раз на спринт) дозволяють уникати накопичення «сюрпризів».

BA організовує зустрічі, на яких:

- ✓ обговорюються нові вимоги,
- ✓ оцінюються їх пріоритети,
- ✓ підтверджується відповідність бізнес-цілям.

Приклад: на спринт-рев'ю замовник просить додати поле «Країна» у форму реєстрації. Це одразу фіксується у backlog.

Висновок. Поєднання versioning, контролю версій та регулярних рев'ю забезпечує:

- ✓ Прозорість процесу змін.
- ✓ Єдине джерело правди (single source of truth).
- ✓ Зменшення ризиків плутанини в документах.

4.6. Оцінка ІТ-проєктів у часі: дисконтування та компаундування

У бізнес-аналітиці час безпосередньо впливає на вартість грошей. Кошти, отримані сьогодні, мають більшу цінність, ніж ті самі кошти, отримані в майбутньому.

Для ІТ-проєктів це критично, адже вони часто потребують значних інвестицій на старті та мають довгострокові вигоди (економія процесів, продаж ліцензій, підписки, монетизація користувачів). Тому використовується два базові фінансові інструменти: дисконтування (discounting) та компаундування (compounding).

4.6.1. Дисконтування (discounting)

Процес приведення майбутніх грошових потоків до їхньої поточної вартості з урахуванням ставки дисконту.

Формула:

$$PV=FV / (1+r)^n$$

де: PV – поточна вартість (Present Value), FV – майбутня вартість (Future Value), r – ставка дисконту (відсоткова ставка або вартість капіталу), n – кількість періодів (років, кварталів, місяців).

Приклад для ІТ-проєкту:

Компанія очікує прибуток від підписки на програмний продукт через 3 роки у розмірі $100 000. Якщо ставка дисконту = 10 %:

$$PV=100000/(1+0.1)3=75131$$

Тобто очікуваний прибуток у $100 000 через 3 роки зараз має «вагу» лише $75 131.

4.6.2. Компаундування (compounding)

Процес нарощення поточної вартості коштів у майбутнє за рахунок відсотків.

Формула:

$$FV=PV\cdot(1+r)^n$$

де: FV – майбутня вартість, PV – поточна вартість, r – ставка зростання, n – кількість періодів.

Приклад для ІТ-проєкту:

Інвестор вклав $50 000 у розробку стартапу. Якщо очікувана дохідність проєкту = 15 % на рік, через 4 роки:

$$FV=50000\cdot(1+0.15)4=87207$$

Отже, початкові інвестиції зростуть до $87 207.

4.6.3. Практичне застосування в бізнес-аналізі ІТ-проєктів

Дисконтування та компаундування застосовуються для того, щоб об'єктивно порівняти витрати, які ми несемо сьогодні, з прибутками, які ми отримаємо в майбутньому. В основі лежить концепція "вартості грошей у часі" (Time Value of Money) – 1000 гривень сьогодні є ціннішими, ніж 1000 гривень через рік (через інфляцію та можливість ці гроші інвестувати).

1. Для дисконтування процес розрахунку сьогоднішньої (поточної) вартості майбутніх грошових потоків, ми ніби "зменшуємо" майбутній прибуток на відсоток інфляції/ризику.

Бізнес-аналітик застосовує дисконтування для:

✓ Техніко-економічне обґрунтування (ТЕО): Коли БА має довести, що проєкт вартий інвестицій.

✓ Розрахуноку NPV (Net Present Value – чиста приведена вартість): Це головне застосування.

Приклад: Ваш ІТ-проєкт (нова CRM-система) коштує $100,000 сьогодні. Ви прогнозуєте, що він принесе економію $50,000 через 1 рік,

$50,000 через 2 роки і $50,000 через 3 роки.

Проблема: Не можна просто додати $50k + $50k + $50k = $150k і сказати, що ми заробили $50k.

Рішення (Дисконтування): ВА бере ставку дисконтування (наприклад, 10 % річних) і розраховує, скільки ці майбутні гроші коштують сьогодні:

$50,000 через 1 рік ≈ $45,454 сьогодні

$50,000 через 2 роки ≈ $41,322 сьогодні

$50,000 через 3 роки ≈ $37,565 сьогодні

Результат: Загальна приведена вартість прибутку = $45,454 + $41,322 + $37,565 = $124,341.

NPV: $124,341 (приведений прибуток) – $100,000 (сьогоднішні витрати) = +$24,341.

Висновок ВА: Проєкт є фінансово доцільним, оскільки його NPV > 0.

Порівняння проєктів: ВА може порівняти два проєкти (наприклад, купити готове рішення чи розробляти своє), розрахувавши NPV для обох варіантів.

2. Компаундування – розрахунок майбутньої вартості сьогоднішніх грошей (зворотний до дисконтування).

Він застосовується бізнес-аналітиклм для:

✓ Оцінка "втраченої вигоди" (Opportunity Cost): Це менш поширений, але важливий аналіз.

Приклад: БА може показати стейкхолдерам: "Якщо ми не інвестуємо ці $100,000 у проєкт, а просто покладемо їх на депозит під 10 % річних, то через 3 роки (завдяки компаундуванню, або складним відсоткам) ми матимемо $133,100."

Це допомагає відповісти на питання: "Чи принесе наш ІТ-проєкт (у приведених грошах) більше, ніж найпростіша банківська інвестиція?"

Підсумок: Дисконтування є критично важливим для бізнес-аналітика, щоб фінансово обґрунтувати ІТ-проєкт, розрахувати його реальну окупність (ROI, NPV) та порівняти різні варіанти інвестицій.

Поєднання обох методів дозволяє обґрунтовано оцінити:

✓ коли проєкт стане прибутковим (break-even point),

✓ яка його реальна теперішня цінність,

✓ наскільки вигідно інвестувати в порівнянні з альтернативами.

У таблиці 4.9 наведено порівняння термінів дисконтування та компаундування.

Таблиця 4.9 – Дисконтування та компаундування

Параметр	Дисконтування (Discounting)	Компаундування (Compounding)
Суть	Приведення майбутніх грошових потоків до їхньої поточної вартості	Нарощення поточної вартості грошей у майбутньому
Формула	$PV = FV / (1+r)^n$	$FV = PV * (1+r)^n$
Питання, яке вирішує	«Скільки майбутні гроші варті зараз?»	«Яку суму сьогоднішні гроші принесуть у майбутньому?»
Приклад в IT-проєкті	Очікуваний дохід від SaaS через 3 роки = $100 000. За ставкою дисконту 10 % поточна вартість = $75 131.	Інвестиція $50 000 у стартап. За ставкою росту 15 % через 4 роки = $87 207.
Практичне застосування	Оцінка NPV, прийняття інвестиційних рішень, аналіз вигідності проєкту	Прогнозування доходів, зростання користувачів, визначення довгострокових вигод
Висновок	Показує «цінність у теперішньому часі»	Показує «зростання у майбутньому»

4.7. Загальні висновки

4.7.1. Роль бізнес-аналітика як ключового мосту між бізнесом і технічними фахівцями

1. Бізнес-аналітик як «перекладач»

✓ Замовники та бізнес-стейкхолдери говорять мовою цілей, прибутків, процесів.

✓ Технічна команда – мовою архітектури, коду, тестів.

✓ Бізнес-аналітик виконує роль «перекладача» між цими двома світами.

Приклад: замовник каже «Хочу, щоб користувачам було зручно», а BA формулює це як конкретну user story: «Система має зберігати останні 5 дій користувача».

2. Комунікаційний міст

BA забезпечує двосторонній діалог:

✓ від бізнесу до команди – чіткі вимоги та пріоритети;

✓ від команди до бізнесу – реалістичні терміни, технічні ризики, обмеження.

✓ Це знижує ризик непорозумінь і помилкових очікувань.

3. Вплив на успіх проєкту

Дослідження PMI показують: більшість провалів у проєктах пов'язана з неякісними вимогами та слабкою комунікацією. Саме BA допомагає уникнути цих ризиків через:

✓ систематизацію вимог,

✓ прозорий процес змін,

✓ управління очікуваннями замовника.

Висновок. Бізнес-аналітик – це не просто «документатор», а стратегічний учасник команди, який поєднує бізнес-бачення із технічною реалізацією (рис.4.20). Його роль як «мосту» робить можливим створення продуктів, які одночасно відповідають потребам користувачів і можливостям технологій.

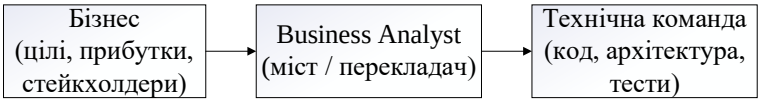

Рисунок 4.20 – BA як міст між бізнесом і технікою

4.7.2. Інструменти збору вимог: від інтерв'ю до прототипів і юзер-сторі

1. Інтерв'ю та опитування.

Один із найпростіших і найефективніших інструментів. Дають змогу

напряму почути думку замовника або кінцевого користувача.

Приклад: інтерв'ю з менеджером з продажів для CRM-системи допомагає виявити критичні функції – збереження контактів та нагадування про дзвінки.

2. Воркшопи та мозкові штурми.

Дають можливість зібрати кількох стейкхолдерів одночасно. Завдяки фасилітації BA узгоджує різні точки зору.

Приклад: на воркшопі для e-commerce обговорюють варіанти фільтрів товарів і домовляються про мінімальний набір.

3. Прототипи.

Візуалізація вимог у вигляді wireframes, mockups або clickable-прототипів. Допомагають клієнту та команді побачити, як виглядатиме продукт.

Приклад: прототип мобільного застосунку дає змогу користувачам перевірити зручність реєстрації ще до розробки.

4. Use Cases.

Формальний опис сценаріїв використання системи. Містять акторів, умови, основний та альтернативні потоки подій.

Приклад: «Користувач оформлює замовлення в інтернет-магазині» – від вибору товару до оплати.

5. User Stories.

Лаконічний спосіб опису вимог у форматі:

«Як [роль], я хочу [дія], щоб [результат]».

Використовуються в Agile, добре підходять для беклогу.

Приклад: «Як покупець, я хочу зберегти товар у «вибране», щоб купити його пізніше».

Висновок Інструменти збору вимог – від інтерв'ю до user stories (рис.4.21) – дають бізнес-аналітику різні рівні деталізації від вербального обговорення до візуалізації та формалізації.

Рисунок 4.21 – Інструменти збору вимог

У комплексі вони дозволяють точніше зрозуміти потреби бізнесу й користувачів.

4.7.3. Трасування – спосіб контролю цілісності вимог

1. Суть трасування.

Трасування вимог (requirements traceability) – це зв'язок між бізнес-вимогами, функціональними вимогами, технічними специфікаціями та тестами. Воно дозволяє контролювати, чи всі вимоги враховані та реалізовані.

2. Навіщо потрібне трасування?

Запобігає втраті вимог. Жодна бізнес-ціль не «загубиться» під час проєкту.

Забезпечує прозорість. Замовник бачить, як його потреби трансформуються у функції продукту.

Підтримує якість. Легко перевірити, чи покрита кожна вимога тестами.

Полегшує аудит. Особливо важливо для регульованих галузей (банківські, медичні, державні системи).

3. Трасування як контроль цілісності.

Якщо у матриці трасування (RTM) є бізнес-вимога, але немає відповідного тесту – це сигнал проблеми. Трасування дозволяє виявляти прогалини (uncovered requirements) і надлишкові функції (gold plating).

Приклад:

Бізнес-вимога: «Користувач має отримувати рахунок після оплати».

Якщо тест-кейс для цієї функції відсутній у RTM → вимога не перевіряється → ризик помилки.

Висновок. Трасування є ключовим механізмом контролю цілісності вимог (рис.4.22):

- ✓ усі цілі бізнесу покриті функціональністю,
- ✓ усі функції перевіряються тестами,
- ✓ зміни прозоро фіксуються у RTM.

Таким чином, BA через трасування гарантує, що кінцевий продукт відповідає як бізнес-очікуванням, так і технічним стандартам.

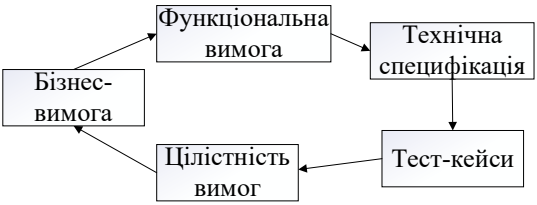

Рисунок 4.22 – Трасування як замкнений цикл

4.7.4. Управління змінами як спосіб збереження актуальності вимог

1. Чому зміни неминучі?

Бізнес-середовище постійно змінюється: конкуренти, технології, закони. Без управління змінами вимоги швидко застарівають, а продукт перестає відповідати очікуванням користувачів.

2. Роль управління змінами:

✓ Забезпечує, щоб вимоги завжди були актуальними.

✓ Допомагає команді залишатися гнучкою, не втрачаючи контроль.

✓ Дає змогу замовнику бачити прозорий процес: що змінюється, чому і як це впливає на строки та бюджет.

3. Інструменти управління змінами

✓ Change Request (CR): офіційний процес внесення змін.

✓ Versioning: відстеження версій вимог.

✓ Регулярні рев'ю: своєчасне узгодження нових потреб із замовником.

✓ Backlog у Agile: постійно оновлюваний список вимог.

4. Переваги:

✓ Підвищення прозорості у відносинах із замовником.

✓ Зменшення ризиків непорозумінь і «забутих» вимог.

✓ Гнучкість без хаосу: зміни проходять контрольований процес.

✓ Продукт відповідає актуальним потребам бізнесу й користувачів.

Висновок. Управління змінами – це механізм збереження актуальності вимог і конкурентоспроможності продукту (рис.4.23). Без цього будь-яка система ризикує стати застарілою вже під час розробки.

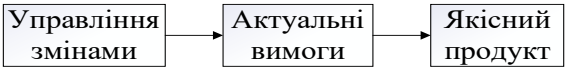

Рисунок 4.23 – Місце Управління змінами у контексті проєкту

4.7.5. Питання для самоперевірки

1. Хто такий бізнес-аналітик і яку роль він відіграє у проєкті?
2. Які основні завдання бізнес-аналітика на різних етапах життєвого циклу вимог?
3. Які soft skills і hard skills є ключовими для BA?
4. Чим відрізняється робота BA у стартапі та у великій корпорації?
5. Які переваги та обмеження методу інтерв'ю для збору вимог?
6. Чим відрізняються відкриті та закриті питання в інтерв'ю?
7. У чому полягає цінність воркшопів і мозкових штурмів для збору вимог?
8. Які переваги використання прототипів на ранніх етапах проєкту?
9. Що таке use case і які основні його елементи?
10. Як user stories допомагають в Agile-командах?
11. Що таке матриця трасування вимог (RTM) і для чого вона використовується?
12. Чому управління змінами у вимогах є критичним для успіху проєкту?

4.7.6. Завдання для самостійної роботи

1. Складіть інтерв'ю для збору вимог: підготуйте 10 відкритих і закритих питань для менеджера з продажів, який буде користуватися CRM-системою.
2. Use Case: опишіть сценарій використання системи «Замовлення товару в інтернет-магазині» (актор, передумови, основний потік, альтернативні потоки).
3. User Stories: складіть 5 user stories для мобільного застосунку для замовлення їжі.
4. Матриця трасування (RTM): побудуйте RTM для простого проєкту «Система бронювання квитків» (мінімум 3 бізнес-вимоги, їх функціональні

реалізації та тест-кейси).

5. Аналіз змін: розробіть приклад Change Request для вимоги «Система має підтримувати оплату карткою» після того, як клієнт попросив додати ще й Apple Pay.

Для завдань 4 і 5 добре підійдуть таблиці (RTM і Change Request).

Шаблон: Матриця трасування вимог (RTM)

ID бізнес-вимоги	Бізнес-вимога	Функціональна вимога	Статус	Тест-кейси

Шаблон: Change Request

ID CR	Автор	Опис зміни	Причина	Вплив	Статус

4.8. Контрольні запитання

1. Що таке бізнес-аналіз у контексті IT-проєктів? Яку роль він відіграє у загальній структурі проєкту?
2. Чому понад 40 % IT-проєктів провалюються через проблеми з вимогами? Як бізнес-аналіз знижує цей ризик?
3. Хто такий бізнес-аналітик (BA)? Опишіть його основні завдання та відповідальність.
4. Яким чином бізнес-аналітик виступає посередником між бізнесом і технічною командою? Наведіть приклад.
5. Які методи збору вимог використовує бізнес-аналітик? Охарактеризуйте інтерв'ю та воркшопи як інструменти.
6. Що таке функціональні та нефункціональні вимоги? У чому різниця та як вони фіксуються?
7. Що таке user story (користувацька історія)? Наведіть приклад та поясніть структуру формату «Як… Я хочу… Щоб…».
8. Що таке модель MoSCoW? Поясніть кожну категорію пріоритизації вимог.

9. Як бізнес-аналіз інтегрується у Waterfall-проєктах і як відрізняється його роль у Agile?
10. Що таке BPMN і UML? Як ці нотації використовуються бізнес-аналітиком для моделювання процесів?
11. Що таке use case (варіант використання) і як він застосовується у бізнес-аналізі?
12. Як бізнес-аналіз допомагає оптимізувати ресурси проєкту і уникнути розробки «непотрібного» функціоналу?
13. Що таке Gap-аналіз і як він застосовується для виявлення розриву між поточним і бажаним станом?
14. Яка роль бізнес-аналітика під час тестування продукту? Як він взаємодіє з QA-командою?
15. Як бізнес-аналітик взаємодіє зі стейкхолдерами протягом усього життєвого циклу проєкту?

5. УПРАВЛІННЯ КОМАНДОЮ ТА КОМУНІКАЦІЯМИ

5.1. Вступ

Значення командної роботи в ІТ-проєктах

1. Чому командна робота важлива?

ІТ-проєкти зазвичай є комплексними: вони включають програмування, тестування, дизайн, аналіз вимог, управління ризиками тощо. Один спеціаліст не може охопити всі напрямки одночасно, тому результат залежить від злагодженої роботи команди. Командна робота дозволяє об'єднати різні компетенції для досягнення єдиної мети – успішного завершення проєкту.

2. Синергія та ефект взаємодії. Синергія означає, що спільна робота дає більший результат, ніж сума індивідуальних зусиль. У команді знання та навички учасників взаємодоповнюються.

Приклад: Розробник створює код, тестувальник знаходить помилки, BA уточнює вимоги, PM координує процес. Разом вони створюють якісний продукт.

3. Взаємозалежність у ІТ-команді

Завдання одного спеціаліста часто впливають на інших. Якщо BA неякісно зібрав вимоги → розробник неправильно реалізує функцію → тестувальник знаходить дефект → строки затримуються. Це показує, що успіх залежить від кожного.

4. Вплив командної роботи на результат

Якісна командна робота = швидше вирішення проблем + краща продуктивність. Команди з сильною культурою співпраці частіше досягають:

- ✓ виконання проєкту у строки;
- ✓ задоволеності замовника;
- ✓ високої якості продукту.

Висновок. Командна робота в ІТ – це не просто сума індивідуальних завдань, а ключовий фактор успіху проєкту (рис.5.1). Вона визначає швидкість розробки, якість кінцевого продукту та задоволеність стейкхолдерів.

Рисунок 5.1 – Командна робота як фактор успіху»

Чому ефективна комунікація впливає на успіх проєкту?

1. Комунікація як основа управління проєктом.

У будь-якому ІТ-проєкті 70-80 % часу менеджера та бізнес-аналітика витрачається на комунікацію (PMI). Вона охоплює всі рівні: від збору вимог до демонстрації готового продукту. Без чіткої, зрозумілої та регулярної комунікації команда втрачає напрям і працює з помилковими очікуваннями.

2. Вплив на строки та якість.

Якщо інформація передається із затримками або спотворюється → зростає ризик затримок і дефектів. Налагоджена комунікація дозволяє швидко вирішувати проблеми та узгоджувати зміни.

Приклад: якщо замовник повідомив про зміну бізнес-процесу вчасно, команда одразу коригує дизайн і розробку, уникаючи переробок.

3. Зменшення ризиків конфліктів.

Багато конфліктів у командах виникають не через різні цілі, а через неправильне розуміння інформації. Ефективна комунікація → прозорість → менше непорозумінь між ролями.

4. Довіра та мотивація.

Регулярні й відкриті комунікації зміцнюють довіру між стейкхолдерами та командою. Коли учасники команди відчувають, що їх чують і розуміють, рівень мотивації зростає.

Висновок. Ефективна комунікація – це кровоносна система ІТ-проєкту (рис.5.2):

- ✓ вона підтримує прозорість,
- ✓ зменшує ризики,
- ✓ забезпечує якість і своєчасність результату.

Рисунок 5.2 – Комунікація як фактор успіху

5.2. Ролі в команді

Основні ролі у команді IT-прорєкту: Розробники, Тестувальники, Бізнес-аналітики, Проєктні менеджери.

5.2.1. Розробники (front-end, back-end, full-stack)

1. Front-end розробники.

Відповідають за користувацький інтерфейс і взаємодію користувача з системою. Використовують технології: HTML, CSS, JavaScript, фреймворки (React, Angular, Vue).

Завдання:

- ✓ створення інтерфейсів, адаптивного дизайну;
- ✓ інтеграція з API;
- ✓ забезпечення зручності та доступності (UX/UI).

Приклад: реалізація особистого кабінету користувача у вебдодатку банку.

2. Back-end розробники

Відповідають за бізнес-логіку та серверну частину системи. Використовують мови та технології: Java, C#, Python, Node.js, PHP, бази даних (PostgreSQL, MySQL, MongoDB).

Завдання:

- ✓ створення API;
- ✓ робота з базами даних;
- ✓ реалізація бізнес-правил;
- ✓ забезпечення безпеки й продуктивності.

Приклад: модуль опрацювання платежів у платіжній системі.

3. Full-stack розробники

Поєднують компетенції front-end і back-end. Гнучкі фахівці, здатні працювати з усіма рівнями додатку. Використовують стек технологій (наприклад, MERN: MongoDB, Express, React, Node.js).

Завдання:

- ✓ швидка розробка прототипів;
- ✓ інтеграція клієнтської та серверної частини;
- ✓ підтримка невеликих команд і стартапів.

Приклад: у стартапі один full-stack інженер може реалізувати MVP продукту.

4. Значення розробників у команді.

Безпосередньо створюють продукт, який бачить користувач. Їхня робота є «кістяком» проєкту (рис.5.3): без написаного коду решта ролей не мають результату для тестування чи аналізу. Співпраця front-end і back-end важлива для узгодженості архітектури та стабільності системи.

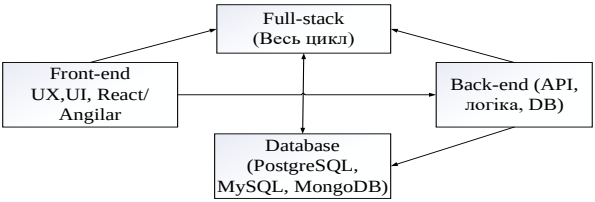

Рисунок 5.3 – Взаємодія розробників

5.2.2. Тестувальники (QA manual, QA automation)

1. Роль тестувальників у команді.

Гарантують, що продукт працює згідно з вимогами та очікуваннями користувачів. Допомагають виявляти дефекти на ранніх етапах, що знижує вартість виправлень. Тестування – це не лише пошук багів, а й перевірка якості, зручності, безпеки.

2. Manual QA (ручне тестування).

Виконує перевірки вручну без автоматизованих інструментів.

Завдання:

- ✓ перевірка інтерфейсів;
- ✓ тестування user flows;

✓ exploratory testing (дослідницьке тестування).

Сильні сторони: гнучкість, інтуїція тестувальника, можливість виявляти «несподівані» проблеми.

Приклад: перевірка процесу оформлення замовлення у вебмагазині вручну.

3. Automation QA (автоматизоване тестування).

Використовує скрипти й фреймворки для автоматизації перевірок.

Завдання:

✓ написання тест-скриптів (Selenium, Cypress, Playwright);

✓ регресійне тестування;

✓ performance-тестування (JMeter, Locust).

Сильні сторони: швидкість, повторюваність, ефективність для великих систем.

Приклад: автоматична перевірка логіну й реєстрації після кожного оновлення системи.

4. Взаємодія Manual і Automation QA.

Обидва підходи доповнюють одне одного:

✓ Manual краще підходить для нових функцій і UX-тестування.

✓ Automation ефективний для повторюваних перевірок і регресійного тестування.

У зрілих командах вони існують паралельно, створюючи збалансований процес забезпечення якості.

5. Значення для проєкту.

Якість продукту = конкурентоспроможність компанії.

Без тестування ризики зростають: користувачі стикаються з багами, падає довіра до продукту (рис.5.4). Тестувальники – «захисники інтересів користувача» в команді.

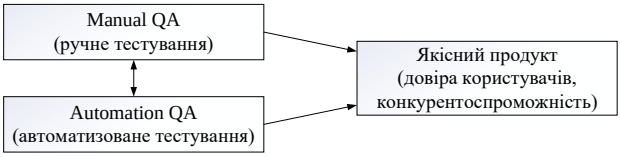

Рисунок 5.4 – Тестувальники

5.2.3. Бізнес-аналітики (Business Analysts, BA)

1. Роль у проєкті.

Бізнес-аналітик – це посередник між бізнесом і технічною командою. Його головне завдання – перетворити бізнес-потреби в зрозумілі та формалізовані вимоги. Забезпечує, щоб кінцевий продукт відповідав як цілям замовника, так і технічним можливостям.

2. Основні завдання:

✓ Збір і документування вимог (інтерв'ю, воркшопи, прототипи, user stories).

✓ Комунікація зі стейкхолдерами, узгодження їхніх очікувань.

✓ Виявлення бізнес-цілей і трансформація їх у технічні рішення.

✓ Підтримка життєвого циклу вимог (traceability, change management).

3. Компетенції BA

✓ Soft skills: комунікація, фасилітація зустрічей, вміння слухати.

✓ Hard skills: моделювання бізнес-процесів (BPMN), UML-діаграми, SQL-запити, аналітичне мислення.

4. Приклади

У стартапі: BA часто виконує кілька ролей – від збору вимог до тестування та координації з клієнтом.

У великій корпорації: BA спеціалізується на окремій сфері (наприклад, робота з фінансовими системами) і є частиною великої аналітичної групи.

5. Значення для проєкту

BA забезпечує єдиний інформаційний простір для бізнесу й команди (рис.5.5). Зменшує ризик невідповідності продукту очікуванням замовника. Є ключовим учасником у процесі управління змінами.

Рисунок 5.5 – BA як міст між бізнесом і технічною командою

5.2.4. Проєктні менеджери (PM)

1. Роль у проєкті.

Проєктний менеджер відповідає за організацію, планування та

контроль виконання проєкту. Його основне завдання – забезпечити, щоб продукт був доставлений у строки, в межах бюджету та з потрібною якістю. PM поєднує управлінські навички з комунікацією та лідерством (рис.5.6).

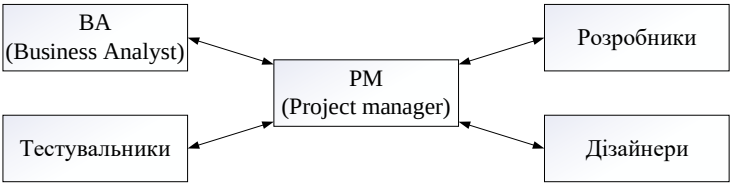

Рисунок 5.6 – PM у центрі взаємодії команди

2. Основні завдання PM

✓ Планування робіт (roadmap, спринти, графік).

✓ Управління ресурсами (команда, бюджет, час).

✓ Координація роботи між BA, розробниками, тестувальниками, дизайнерами.

✓ Управління ризиками.

✓ Звітність перед замовником і внутрішнім керівництвом.

3. Компетенції PM

✓ Soft skills: лідерство, вирішення конфліктів, фасилітація зустрічей, навички переконання.

✓ Hard skills: володіння методологіями (Agile, Scrum, Kanban, Waterfall, PMBOK), інструментами (Jira, MS Project, Trello).

4. Значення для проєкту

PM – це «капітан корабля», який задає напрямок руху команди. Від його здатності координувати залежить швидкість і якість досягнення результатів (рис.5.6). Ефективний менеджер створює умови, у яких команда може працювати максимально продуктивно.

5. Приклади

У стартапі: PM може одночасно виконувати функції BA і координатора.

У великій корпорації: PM керує кількома командами, веде масштабні проєкти та відповідає за стратегічне узгодження.

5.2.5. Допоміжні ролі: UX/UI дизайнер, DevOps, архітектор, Product Owner

1. UX/UI дизайнер

Відповідає за зовнішній вигляд та зручність використання продукту.

Завдання:

- ✓ створення прототипів і макетів;
- ✓ дослідження потреб користувачів;
- ✓ забезпечення доступності (accessibility).

Інструменти: Figma, Sketch, Adobe XD.

Приклад: дизайн мобільного застосунку з урахуванням принципів user-centered design.

2. DevOps інженер.

Поєднує розробку (Dev) і операційні процеси (Ops).

Завдання:

- ✓ автоматизація CI/CD (continuous integration / continuous delivery);
- ✓ моніторинг та підтримка інфраструктури;
- ✓ забезпечення безперебійної роботи системи.

Інструменти: Docker, Kubernetes, Jenkins, AWS, Azure.

Приклад: автоматичний деплой нових версій застосунку на сервери без простоїв.

3. Архітектор.

Відповідає за високорівневу структуру системи.

Завдання:

- ✓ вибір технологічного стеку;
- ✓ проєктування архітектури (моноліт, мікросервіси, serverless);
- ✓ забезпечення масштабованості та безпеки.

Приклад: визначення, чи буде платформа e-commerce реалізована як моноліт чи набір мікросервісів.

4. Product Owner (PO)

Представник бізнесу в команді розробки (часто в Agile-проєктах).

Завдання:

- ✓ формування та пріоритизація беклогу;
- ✓ визначення цінності функціоналу для бізнесу;

✓ участь у плануванні та рев'ю.

Приклад: PO вирішує, які фічі потраплять у наступний спринт на основі відгуків користувачів.

5. Значення допоміжних ролей

Ці ролі не завжди присутні в маленьких командах, але у великих проєктах їх внесок є критичним. UX/UI дизайнери відповідають за user experience, DevOps – за стабільність і швидкість релізів, архітектори – за структурну якість, Product Owner – за цінність для бізнесу. Разом вони підсилюють основну команду й підвищують шанси проєкту на успіх (рис.5.7).

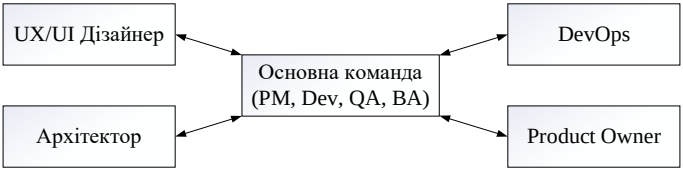

Рисунок 5.7 – Допоміжні ролі навколо основної команди

5.2.6. Взаємодія між ролями: як формується баланс у команді

1. Суть взаємодії

IT-команда складається з фахівців різного профілю: Dev, QA, BA, PM, дизайнерів, DevOps тощо. Кожна роль має свою зону відповідальності, але успіх залежить від їхньої співпраці. Якщо одна роль відсутня або неефективна, страждає вся система (наприклад, відсутність QA → більше дефектів у продакшн).

2. Принципи ефективної взаємодії:

✓ Прозорість: усі розуміють завдання та цілі.

✓ Взаємоповага: цінується внесок кожного фахівця.

✓ Спільна відповідальність: результат – командний, а не індивідуальний.

✓ Регулярна комунікація: мітинги, рев'ю, ретроспективи.

3. Приклади взаємодії

✓ BA + PM: BA формалізує вимоги, PM перетворює їх у план.

✓ Dev + QA: Dev створює код, QA перевіряє й допомагає уникати дефектів.

✓ UX/UI дизайнер + Dev: дизайнер формує прототипи, Dev реалізує їх у коді.

✓ DevOps + Dev: DevOps забезпечує стабільність інфраструктури, Dev – якість коду.

4. Баланс у команді

Баланс (рис.5.8) досягається тоді, коли немає “слабкої ланки”. Якщо хтось з ролей домінує, а інші знецінюються → виникають конфлікти. Кращий результат досягається, коли кожна роль:

✓ чітко знає свої обов’язки;

✓ цінує внесок інших;

✓ працює на спільну мету.

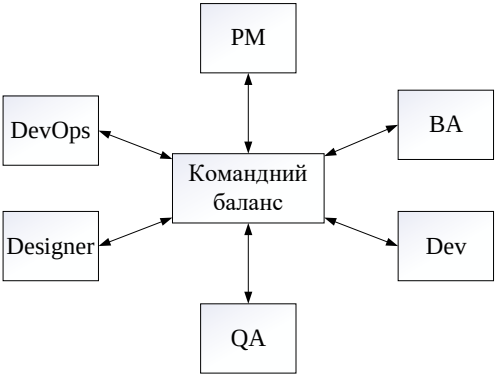

Рисунок 5.8 – Командний баланс через взаємодію ролей

5.2.7. Приклади організації команд у стартапах і корпоративних проєктах

1. Команда у стартапі.

Характеристика: обмежений бюджет, швидкі зміни, орієнтація на MVP (minimum viable product).

Склад команди: зазвичай невеликий (3-7 людей):

✓ 1-2 full-stack розробники;

- ✓ QA (часто ручне тестування або виконують інші члени команди);
- ✓ UX/UI дизайнер (може бути суміщений з front-end);
- ✓ BA або PM (іноді засновник виконує цю роль).

Особливості:

- ✓ мультифункціональність – одна людина виконує кілька ролей;
- ✓ акцент на швидкості та гнучкості;
- ✓ мінімум формальних процесів.

Приклад: команда зі стартапу в сфері EdTech – один full-stack інженер створює MVP, дизайнер одночасно займається маркетинговими матеріалами.

2. Команда у великій корпорації.

Характеристика: великий бюджет, складні та довготривалі проєкти, сувора формалізація.

Склад команди: може включати десятки або сотні фахівців, організованих у підкоманди (squads, tribes):

- ✓ окремі front-end і back-end команди;
- ✓ кілька QA команд (manual і automation);
- ✓ бізнес-аналітики за напрямами;
- ✓ кілька PM або Program Manager;
- ✓ DevOps, архітектори, спеціалізовані дизайнери.

Особливості:

- ✓ висока спеціалізація ролей;
- ✓ застосування стандартів (PMBOK, ITIL, ISO);
- ✓ чітка ієрархія та система звітності.

Приклад: банк запускає мобільний додаток – окремі команди відповідають за безпеку, інтеграцію з платіжними системами, мобільний UX, бекенд і тестування.

Висновок.

У стартапах важлива гнучкість і універсальність.

У корпораціях – стабільність, масштабованість і стандартизація.

Жоден підхід не є «кращим», усе залежить від контексту: швидке тестування ідеї чи довгострокова підтримка складної системи (табл.5.1).

Таблиця 5.1 – Порівняння стартапу та корпорації

Ознака	Стартап	Корпорація
Розмір команди	3-7 людей	десятки й сотні
Ролі	мультифункціональні	вузькоспеціалізовані
Фокус	швидкість, MVP	якість, масштабованість
Методологія	Agile, Lean Startup	Waterfall + Agile (гібрид)
Процеси	мінімум документації	сувора формалізація

5.3. Динаміка розвитку команди

При відсутності розвитку проходження всіх етапів прорєктування у ІТ-сфері стає дуже складною задачею.

5.3.1. Модель Брюса Такмана (Tuckman’s stages of group development)

Модель Брюса Такмана описує закономірності розвитку команд та етапи, які проходить будь-яка група людей у процесі становлення та спільної роботи. Вперше модель була представлена у 1965 році у статті "Developmental Sequence in Small Groups".

Вона отримала широке застосування у сфері управління проєктами, HR та психології командної роботи, оскільки допомагає керівникам та учасникам команд усвідомити природні труднощі групової динаміки та ефективно управляти ними.

Основні етапи моделі (Tuckman’s stages):

1. Forming (Формування)

✓ Команда тільки створюється.

✓ Учасники обережні, намагаються зрозуміти завдання, ролі, очікування.

✓ Рівень продуктивності низький, важливіше – знайомство та встановлення контактів.

2. Storming (Конфлікт, шторм)

✓ Виникають перші суперечності, боротьба за ролі та лідерство.

✓ Можливі конфлікти через різні підходи до роботи.

✓ Завдання керівника – допомогти конструктивно вирішити

конфлікти та спрямувати енергію у продуктивне русло.

3. Norming (Нормування)

✓ Команда виробляє правила взаємодії, формуються норми та спільні цінності.

✓ Зростає довіра, учасники починають підтримувати один одного.

✓ З'являється чіткіше розуміння ролей та завдань.

4. Performing (Виконання)

✓ Команда стає зрілою та продуктивною.

✓ Учасники ефективно співпрацюють, зосереджені на результаті.

✓ Конфлікти вирішуються швидко й конструктивно, рівень автономії високий.

5. Adjourning (Завершення, розформування) (додано у 1977 р.)

✓ Етап розпуску команди після виконання проєкту.

✓ Учасники можуть відчувати сум, втому або гордість за результат.

✓ Важливо підвести підсумки, відзначити досягнення та забезпечити плавний перехід до нових завдань.

Значення моделі:

✓ Дає розуміння природної динаміки розвитку команд.

✓ Допомагає менеджерам і лідерам прогнозувати проблеми на кожному етапі.

✓ Сприяє формуванню ефективної командної культури.

✓ Використовується в Agile, Scrum, HR-практиках, проєктному менеджменті.

5.3.1.1. Модель Брюса Такмана – Forming (формування)

1. Суть етапу.

Це початкова стадія, коли команда тільки формується. Учасники ще не знають одне одного, шукають своє місце та ролі. Спостерігається ввічливість, обережність, орієнтація на лідера.

2. Характерні ознаки:

✓ Невизначеність у розподілі ролей.

✓ Залежність від керівника (PM або Team Lead).

✓ Орієнтація на правила й очікування.

✓ Мінімальна конфліктність (бо люди ще не показують справжніх емоцій).

3. Завдання керівника на цьому етапі:

✓ Чітко визначити цілі проєкту та команди.

✓ Пояснити ролі та відповідальність.

✓ Створити безпечне середовище для знайомства.

✓ Забезпечити первинну мотивацію.

4. Приклад

Нова команда у стартапі тільки зібралася для створення мобільного застосунку. PM проводить kick-off meeting: представляє учасників, озвучує мету MVP, знайомить з процесами.

Висновок. Етап Forming (рис.5.9)- це фундамент. Від того, наскільки добре команда пройде цей етап, залежить ефективність наступних фаз (Storming, Norming, Performing).

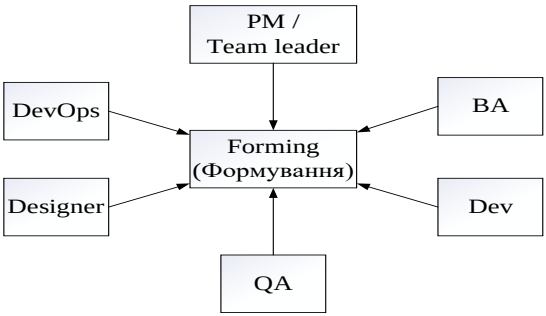

Рисунок 5.9 – Етап Forming

*5.3.1.2. Модель Брюса Такмана – Storming (шторм*ін*г)*

1. Суть етапу.

Це друга стадія розвитку команди. Після початкової ввічливості учасники починають відстоювати свої позиції, проявляти емоції та конфліктувати. Це природний процес, адже формується справжня динаміка взаємодії.

2. Характерні ознаки

✓ Конфлікти щодо розподілу ролей і зон відповідальності.

✓ Сумніви в цілях і завданнях.

✓ Розбіжності у стилях роботи.

✓ Опір правилам або рішенням лідера.

✓ Можливі «підгрупи» всередині команди.

3. Завдання керівника на цьому етапі:

✓ Бути медіатором і допомагати вирішувати конфлікти конструктивно.

✓ Забезпечувати прозору комунікацію.

✓ Допомагати команді сфокусуватися на спільних цілях.

✓ Заохочувати відкритість та повагу до думок інших.

4. Приклад

У команді з'являються суперечки між розробниками та QA щодо пріоритетів виправлення багів. PM організовує спільний воркшоп, де разом визначають критерії пріоритетності.

Висновок. Етап Storming (рис.5.10) неминучий і навіть корисний. Якщо команда успішно його проходить, зростає рівень довіри та зрілості. Якщо ж конфлікти не вирішуються, команда ризикує «застрягти» й не перейти на наступний рівень.

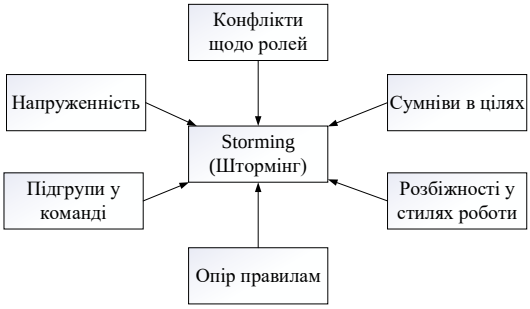

Рисунок 5.10 – Етап Storming

5.3.1.3. Модель Брюса Такмана – Norming (нормування)

1. Суть етапу

Це третя стадія розвитку команди. Після періоду конфліктів учасники починають домовлятися, виробляти правила та цінності спільної роботи. Формується довіра та командна згуртованість.

2. Характерні ознаки

✓ Зменшення конфліктності, зростання співпраці.

✓ Вироблення спільних норм і стандартів.

✓ Учасники приймають свої ролі.

✓ Зростає відчуття «ми» замість «я».

✓ Поступове підвищення продуктивності.

3. Завдання керівника

✓ Підтримувати і закріплювати позитивну динаміку.

✓ Заохочувати обмін знаннями та взаємодопомогу.

✓ Допомагати формувати командні правила (кодекс, Definition of Done).

✓ Передавати частину відповідальності самій команді.

4. Приклад

Команда після кількох суперечок домовляється про правила комунікації:

✓ завжди писати статус у Jira,

✓ обговорювати зміни на щоденних мітингах,

✓ дотримуватися Code Style.

Висновок. Етап Norming (рис.5.11) – це точка, де команда переходить від конфліктів до співпраці. Зростає довіра, взаємна повага й орієнтація на результат. Це фундамент для досягнення найвищої продуктивності на наступному етапі – Performing.

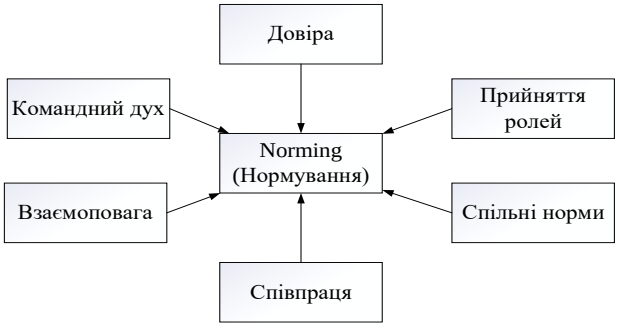

Рисунок 5.11- Етап Norming,

5.3.1.4. Модель Брюса Такмана – Performing (ефективна робота)

1. Суть етапу

Це четверта стадія розвитку команди, на якій вона досягає максимальної продуктивності. Учасники добре знають свої ролі, довіряють один одному та працюють злагоджено. Лідер (PM/Team Lead) може делегувати більшість рішень, адже команда стає самоорганізованою.

2. Характерні ознаки

- ✓ Висока ефективність і продуктивність.
- ✓ Самостійне вирішення більшості проблем.
- ✓ Мінімум конфліктів, які швидко розв'язуються.
- ✓ Висока мотивація та командний дух.
- ✓ Фокус на результат і цінність для замовника.

3. Завдання керівника

- ✓ Виконувати роль коуча й фасилітатора, а не контролера.
- ✓ Забезпечувати команду ресурсами.
- ✓ Стежити за стратегічними цілями та пріоритетами.
- ✓ Допомагати у вирішенні складних зовнішніх питань.

4. Приклад

Команда працює над великим релізом: Dev, QA, BA і дизайнер самостійно узгоджують деталі, а PM лише координує комунікацію зі стейкхолдерами. Завдання виконуються вчасно, якість висока, замовник задоволений (рис.5.12).

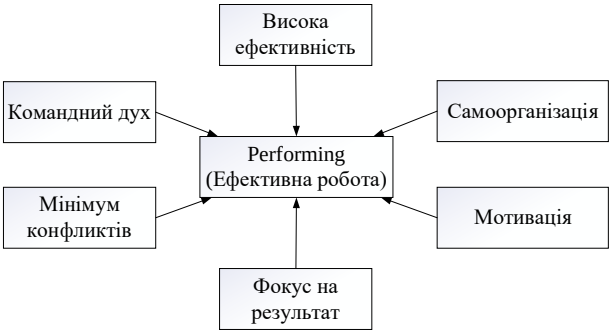

Рисунок 5.12 – Етап Performing

Висновок. Етап Performing – це найвищий рівень зрілості команди. Досягнувши його, команда здатна працювати автономно, швидко адаптуватися та приносити максимальну користь бізнесу.

5.3.1.5. Модель Брюса Такмана – Adjourning (завершення)

1. Суть етапу

Це п’ята стадія розвитку команди, яку Такман додав пізніше. Вона настає тоді, коли команда виконала свої завдання і проєкт завершується. Основна особливість – розформування команди та перехід учасників до нових проєктів.

2. Характерні ознаки

✓ Відчуття втрати (особливо якщо команда працювала довго).

✓ Зниження мотивації наприкінці, бо учасники думають про майбутнє.

✓ Потреба у підбитті підсумків і визнанні результатів.

✓ Можливість стресу чи розчарування через розпад команди.

3. Завдання керівника

✓ Організувати фінальний огляд результатів (project review, retrospective).

✓ Визнати внесок кожного учасника, подякувати за роботу.

✓ Допомогти команді плавно перейти до нових проєктів.

✓ Заохотити збереження контактів і знань (knowledge transfer).

4. Приклад

Проєкт у банківській сфері завершився після 2 років. PM організував фінальний воркшоп, де команда обговорила досягнення, проблеми та уроки. Кожному члену видали сертифікати подяки, а знання були задокументовані у Confluence.

Висновок. Етап Adjourning (рис.5.13) важливий для закриття команди так само, як Forming для її старту. Він дозволяє:

✓ підбити підсумки,

✓ зберегти знання,

✓ підтримати моральний стан людей.

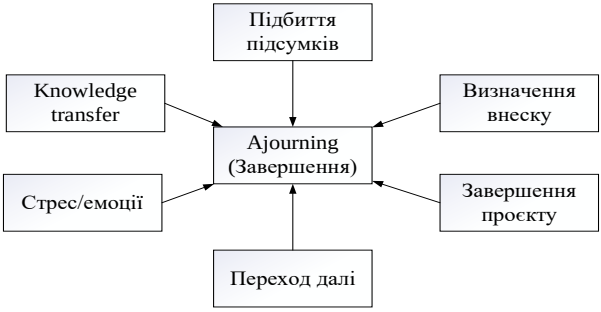

Рисунок 5.13 – Етап Adjourning.

5.3.1.6. Характеристики кожного етапу моделі Такмана

1. Forming (Формування)

✓ Учасники ще не знають один одного.

✓ Переважає ввічливість та обережність у висловлюваннях.

✓ Високий рівень залежності від лідера.

✓ Завдання: знайомство, визначення цілей і ролей.

2. Storming (Штормінг)

✓ Виникають перші конфлікти й суперечки.

✓ Можливий опір правилам або стилю управління.

✓ Учасники намагаються відстояти власні позиції.

✓ Завдання: навчитися конструктивно вирішувати конфлікти, сформувати довіру.

3. Norming (Нормування)

✓ Зменшується рівень конфліктності.

✓ Формуються спільні правила, стандарти та норми.

✓ Зростає відчуття командної ідентичності («ми» замість «я»).

✓ Завдання: закріпити командні норми, виробити сталі процеси взаємодії.

4. Performing (Ефективна робота)

✓ Команда працює максимально продуктивно.

✓ Самоорганізація та самостійне прийняття рішень.

✓ Висока мотивація та командний дух.

✓ Завдання: досягати бізнес-цілей проєкту з найвищою ефективністю.

5. Adjourning (Завершення)

✓ Команда виконує цілі та розпадається.

✓ Можливі змішані емоції: гордість за результат і сум через розформування.

✓ Важлива передача знань та підбиття підсумків.

✓ Завдання: завершити роботу, визнати внесок кожного, підготувати учасників до нових викликів.

Усі етапи моделі Такмана зображені на рисунку 5.14.

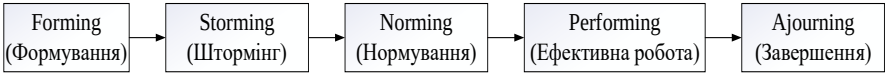

Рисунок 5.14 – Усі етапи моделі Такмана

5.3.2. Роль PM та BA на різних стадіях розвитку команди

1. Forming (Формування)

PM (Project Manager):

✓ Організовує kick-off meeting.

✓ Чітко визначає цілі та завдання.

✓ Встановлює правила комунікації.

✓ Створює атмосферу довіри.

BA (Business Analyst):

✓ Допомагає уточнити бізнес-цілі.

✓ Перекладає вимоги клієнта у зрозумілу форму для команди.

✓ Пояснює бізнес-контекст.

2. Storming (Штормінг)

PM:

✓ Виконує роль фасилітатора й медіатора у конфліктах.

✓ Допомагає вирішувати суперечки конструктивно.

✓ Нагадує про спільні цілі.

BA:

✓ Допомагає зняти непорозуміння щодо вимог.

✓ Консультує команду про бізнес-пріоритети.

✓ Забезпечує прозорість вимог, щоб зменшити конфлікти.

3. Norming (Нормування)

PM:

✓ Підтримує формування командних правил.

✓ Делегує частину відповідальності.

✓ Заохочує взаємодопомогу й розвиток довіри.

BA:

✓ Допомагає створити спільний backlog.

✓ Формалізує процеси збору й оновлення вимог.

✓ Сприяє узгодженості між бізнес-очікуваннями й командною роботою.

4. Performing (Ефективна робота)

PM:

✓ Виконує роль коуча й стратега.

✓ Мінімізує мікроменеджмент, довіряє команді.

✓ Фокусується на стратегічних питаннях і стейкхолдерах.

BA:

✓ Оптимізує роботу з вимогами.

✓ Допомагає команді бачити бізнес-цінність у кожній задачі.

✓ Працює з клієнтом над стратегічними ініціативами.

5. Adjourning (Завершення)

PM:

✓ Організовує ретроспективу й підбиття підсумків.

✓ Відзначає досягнення команди.

✓ Допомагає учасникам плавно перейти до нових проєктів.

BA:

✓ Формує фінальну документацію.

✓ Передає знання майбутнім командам.

✓ Узагальнює lessons learned щодо вимог і процесів.

Порівняння ролів Project Manager та Business Analyst наведено у таблиці 5.2.

Таблиця 5.2 – Порівняльна таблиця ролів PM та BA

Етап	PM (Project Manager)	BA (Business Analyst)
Forming (Формування)	- Організовує kick-off meeting. - Визначає цілі та завдання. - Встановлює правила комунікації. - Створює атмосферу довіри.	- Уточнює бізнес-цілі. - Перекладає вимоги клієнта у зрозумілу форму. - Пояснює бізнес-контекст.
Storming (Штормінг)	- Виконує роль фасилітатора й медіатора. - Допомагає вирішувати суперечки конструктивно. - Нагадує про спільні цілі.	- Допомагає зняти непорозуміння щодо вимог. - Консультує про бізнес-пріоритети. - Забезпечує прозорість вимог.
Norming (Нормування)	- Підтримує формування правил. - Делегує частину відповідальності. - Заохочує взаємодопомогу.	- Допомагає створити backlog. - Формалізує процеси збору й оновлення вимог. - Сприяє узгодженості очікувань.
Performing (Ефективна робота)	- Виконує роль коуча й стратега. - Мінімізує мікроменеджмент. - Фокусується на стратегічних питаннях.	- Оптимізує роботу з вимогами. - Допомагає бачити бізнес-цінність у задачах. - Працює над стратегічними ініціативами.
Adjourning (Завершення)	- Організовує ретроспективу. - Відзначає досягнення команди. - Допомагає плавно перейти до нових проєктів.	- Формує фінальну документацію. - Передає знання наступним командам. - Узагальнює lessons learned.

5.3.3. Приклади з реальних ІТ-команд

1. Forming (Формування)

Приклад: Компанія запускає стартап у сфері FinTech.

✓ Зібрали команду з 5 осіб: 2 розробники, 1 дизайнер, 1 QA і 1 PM.

✓ На kick-off meeting усі знайомляться, обговорюють бачення продукту.

✓ Більшість рішень приймає PM, команда ще не впевнена у своїй взаємодії.

2. Storming (Штормінг)

Приклад: Під час роботи над MVP розробники й QA сперечаються:

✓ QA вимагає більше часу на тестування.

✓ Розробники хочуть швидше показати продукт інвесторам.

✓ PM проводить воркшоп з пріоритизації багів, щоб знизити напругу.

3. Norming (Нормування)

Приклад: Через кілька спринтів команда домовляється:

✓ Використовувати Definition of Done (завдання вважається завершеним лише після проходження тестів).

✓ Статуси задач оновлюються в Jira.

✓ Конфлікти зменшуються, команда стає згуртованішою.

4. Performing (Ефективна робота)

Приклад: У зрілій продуктовій компанії команда з 12 людей (Dev, QA, BA, дизайнер, DevOps) працює самостійно.

✓ Вони узгоджують архітектурні рішення без PM.

✓ PM лише комунікує із замовником і трекає високорівневі цілі.

✓ Команда регулярно видає стабільні релізи.

5. Adjourning (Завершення)

Приклад: Після 1,5 року команда завершує проєкт з eCommerce-платформою.

✓ PM організовує фінальну ретроспективу.

✓ Учасники діляться досвідом, формують lessons learned.

✓ BA оформлює документацію для наступної команди підтримки.

✓ Команду розформовують, частину учасників переводять на нові проєкти.

Висновок. Модель Такмана підтверджується в реальних IT-командах.

Усі етапи – від знайомства до розпаду – проходять навіть невеликі стартапи й великі корпорації.

Головна роль PM та BA – допомогти команді безболісно пройти всі стадії та зберегти ефективність.

5.4. Ефективна комунікація зі стейкхолдерами

Стейкхолдери – це всі особи або групи, які мають інтерес у проєкті, можуть впливати на нього або зазнавати впливу від його результатів. Вони не завжди безпосередньо беруть участь у роботі над продуктом, але їхні очікування та потреби впливають на цілі й хід проєкту.

5.4.1. Характеристики стейкхолдери та їх значення

1. Типи стейкхолдерів

✓ Внутрішні: керівництво компанії, проєктна команда, суміжні відділи.

✓ Зовнішні: клієнти, інвестори, постачальники, партнери, регулятори.

✓ Первинні (ключові): ті, хто напряму зацікавлений у результатах (замовник, кінцеві користувачі).

✓ Вторинні: ті, хто опосередковано залучений (державні органи, ЗМІ, суспільство).

2. Чому стейкхолдери важливі

✓ Визначають успіх або провал проєкту: навіть ідеальне технічне рішення може вважатися невдалим, якщо воно не відповідає очікуванням ключових стейкхолдерів.

✓ Допомагають уточнити вимоги та пріоритети.

✓ Можуть стати джерелом ризиків (наприклад, зміна вимог інвестора).

✓ Формують репутацію команди та компанії.

3. Приклад з ІТ-практики. У проєкті зі створення банківського мобільного застосунку:

✓ Стейкхолдери: банк (замовник), клієнти банку (користувачі), НБУ (регулятор).

✓ Якщо команда не врахує вимог НБУ, продукт не буде допущений до використання, навіть якщо замовнику та користувачам він подобається.

Висновок. Розуміння того, хто є стейкхолдерами, – це перший і найважливіший крок до ефективної комунікації (рис.5.15). PM та BA повинні вміти ідентифікувати стейкхолдерів, аналізувати їхні інтереси та формувати комунікаційну стратегію.

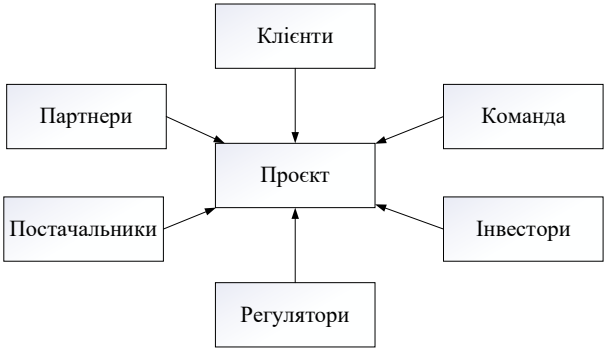

Рисунок 5.15 – Стейкхолдери в центрі проєкту

5.4.2. Типи комунікацій зі стейкхолдерами

1. За напрямком:

✓ Вертикальна комунікація – між керівництвом і командою (PM ↔ топ-менеджмент).

✓ Горизонтальна – між рівними за статусом учасниками (Dev ↔ QA, BA ↔ Designer).

✓ Діагональна – між різними рівнями та підрозділами (наприклад, інвестор ↔ розробник під час демо).

2. За форматом (табл.5.3):

✓ Формальна: звітність, офіційні презентації, контрактні переговори.

✓ Неформальна: розмови на каві, чати в месенджерах, короткі обговорення.

3. За каналами (рис.5.16):

✓ Очні зустрічі (мітинги, воркшопи).

✓ Онлайн-зустрічі (Zoom, Teams, Meet).

✓ Письмова комунікація (електронна пошта, Confluence, Jira).

✓ Миттєві повідомлення (Slack, Telegram).

4. За частотою:

✓ Регулярна: щоденні стендапи, щотижневі синки, щомісячні Steering Committee.

✓ Ad-hoc: комунікація за потреби (екстрені дзвінки, повідомлення про критичні баги).

5. Приклад з ІТ-практики

У великому eCommerce-проєкті:

✓ Формальні канали: щотижневі звіти для замовника, презентації релізів.

✓ Неформальні: чат у Slack для швидких обговорень.

✓ Вертикальні: PM звітує CEO замовника.

✓ Горизонтальні: співпраця Dev ↔ QA у Jira.

Таблиця 5.3 – Формальна vs Неформальна комунікація

Формальна	Неформальна
Звіти, презентації, офіційні документи	Розмови під час кави, швидкі чати
Контрактні переговори	Емоційні реакції у месенджерах
Регулярні планові зустрічі	Ad-hoc обговорення проблем

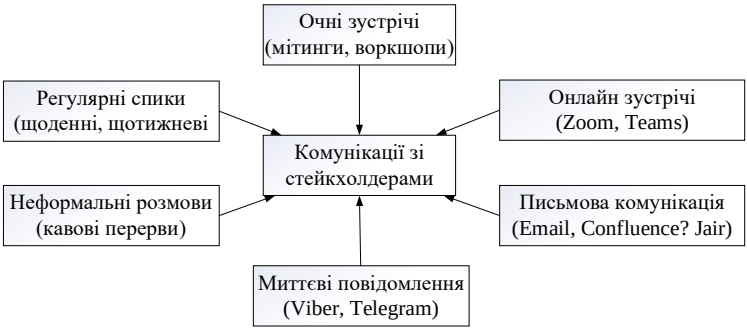

Рисунок 5.16 – Канали комунікацій

5.4.3. Інструменти комунікації (Jira, Confluence, Slack, Zoom)

1. Jira

Інструмент для управління задачами та проєктами. Використовується для створення backlog, user stories, баг-репортів. Дає можливість прозоро відслідковувати статус роботи.

Приклад: замовник може бачити прогрес по кожній задачі в режимі реального часу.

2. Confluence

Корпоративна wiki-система для збереження знань. Використовується для документації вимог, технічних специфікацій, протоколів зустрічей. Допомагає уникнути втрати інформації.

Приклад: протокол воркшопу зі стейкхолдерами зберігається у Confluence, і кожен має доступ у будь-який момент.

3. Slack

Чат-платформа для швидкої комунікації. Підтримує канали, інтеграції (з Jira, GitHub, CI/CD). Добре підходить для щоденного обговорення, але не замінює офіційних документів.

Приклад: окремий канал для клієнта, де команда швидко відповідає на питання.

4. Zoom

Інструмент для відеозв'язку й онлайн-зустрічей. Використовується для демо, щотижневих синків, воркшопів. Дозволяє залучати віддалених стейкхолдерів у реальному часі.

Приклад: демо нового релізу для клієнтів з різних країн.

Висновок. Ефективна комунікація неможлива без поєднання кількох інструментів:

- ✓ Jira + Confluence = прозора документація й управління задачами.
- ✓ Slack + Zoom = швидка взаємодія та живе спілкування.

Разом вони створюють екосистему (рис.5.17), яка дозволяє враховувати інтереси стейкхолдерів і мінімізувати ризики непорозумінь.

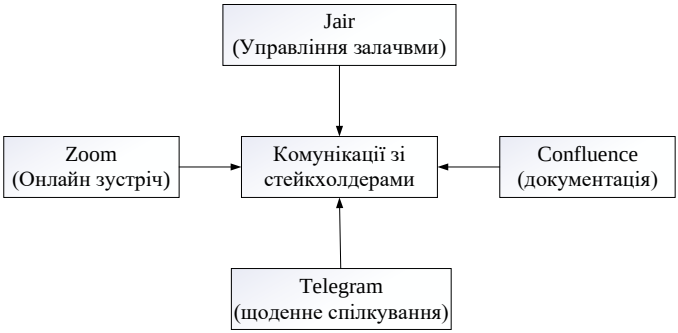

Рисунок 5.17 – Екосистема комунікацій.

5.4.4. Вибір каналу комунікації залежно від типу стейкхолдера

1. Топ-менеджмент / Замовники

✓ Канали: офіційні презентації, звіти, Zoom-зустрічі, Email.

✓ Причина: вони цікавляться результатами та стратегічними рішеннями, а не деталями.

✓ Формат: коротко, по суті, з акцентом на бізнес-цінність.

2. Кінцеві користувачі

✓ Канали: опитування, інтерв'ю, юзабіліті-тести, демо-презентації.

✓ Причина: потрібно отримати зворотний зв'язок про продукт, зрозуміти потреби.

✓ Формат: зрозумілою мовою, без зайвої технічної термінології.

3. Проєктна команда

✓ Канали: Slack/Teams для щоденного спілкування, Jira для задач, Confluence для документації, стендапи.

✓ Причина: необхідна швидка й постійна комунікація.

✓ Формат: детально, технічно, з можливістю швидкого уточнення.

4. Регулятори / Державні органи

✓ Канали: офіційні документи, формальні звіти, юридично вивірена комунікація.

✓ Причина: потрібна чітка відповідність законам і стандартам.

✓ Формат: максимально формально, без двозначностей.

5. Інвестори

✓ Канали: дашборди з метриками, щомісячні/щоквартальні звіти, презентації у Zoom.

✓ Причина: вони очікують прозорість щодо фінансових і бізнес-показників.

✓ Формат: цифри, KPI, ROI, прогноз прибутків.

6. Постачальники / Партнери

✓ Канали: Email, контрактні переговори, робочі зустрічі в Zoom.

✓ Причина: важлива синхронізація графіків і відповідальності.

✓ Формат: формально, але з простором для уточнень.

Висновок. Вибір каналу залежить від:

✓ рівня залученості стейкхолдера,

- ✓ його ролі у проєкті,
- ✓ формальності відносин.

PM і BA мають вміти підбирати правильний канал і формат комунікації, щоб уникати непорозумінь і підвищувати ефективність проєкту (табл.5.4).

Таблиця 5.4 – Тип стейкхолдера → Канали комунікації → Формат

Тип стейкхолдера	Канали комунікації	Формат
Топ-менеджмент / Замовники	Офіційні презентації, звіти, Zoom-зустрічі, Email	Коротко, по суті, акцент на бізнес-цінність
Кінцеві користувачі	Опитування, інтерв'ю, юзабіліті-тести, демо-презентації	Простою мовою, без технічних термінів
Проєктна команда	Slack/Teams, Jira, Confluence, стендапи	Детально, технічно, з можливістю швидких уточнень
Регулятори / Державні органи	Офіційні документи, формальні звіти	Максимально формально, юридично вивірено
Інвестори	Дашборди, щомісячні/щоквартальні звіти, Zoom-презентації	Цифри, KPI, ROI, прогнози
Постачальники / Партнери	Email, контрактні переговори, Zoom-зустрічі	Формально, але з можливістю уточнень

5.4.5. Комунікаційний план: як, коли і з ким

Комунікаційний план – це документ, який визначає, хто, коли, як і з якою метою отримує інформацію про проєкт, часто є частиною Project Management Plan у рамках стандартів PMBOK та дозволяє уникати інформаційних прогалин і дублювання (рис.5.18).

Основні елементи (табл 5.5):

- ✓ Адресати (стейкхолдери): хто потребує інформації.
- ✓ Тип інформації: статус проєкту, ризики, рішення, прогрес задач.
- ✓ Формат і канал: звіти, презентації, стендапи, чати.

✓ Частота: щодня, щотижня, щомісяця, ad-hoc.

✓ Відповідальні: хто готує й відправляє інформацію (PM, BA, Team Lead).

4. Переваги комунікаційного плану:

✓ Забезпечує прозорість і довіру між усіма сторонами.

✓ Знижує ризик непорозумінь.

✓ Допомагає зберегти баланс між формальною та неформальною комунікацією.

✓ Створює основу для ефективного управління стейкхолдерами.

Рисунок 5.18 – Комунікаційний план

Таблиця 5.5 – Приклад структури комунікаційного плану

Стейкхолдер	Інформація	Формат/Канал	Частота	Відповідальний
Замовник	Прогрес проєкту, ризики	Звіт у PDF + Zoom-зустріч	Щотижня	PM
Команда	Статус задач	Daily stand-up у Slack/Zoom	Щодня	Team Lead
Інвестори	KPI, ROI, фінансові прогнози	Презентація + дашборд	Щоквартально	PM + BA
Регулятори	Відповідність вимогам	Офіційні документи	За вимогою	BA

5.4.6. Приклади хороших і поганих комунікацій

Хороші приклади

1. Регулярні оновлення замовнику

✓ PM щотижня надсилає звіт + проводить коротку Zoom-зустріч.

✓ Замовник завжди знає статус проєкту, довіра зростає.

2. Прозора документація

✓ BA веде Confluence: вимоги, рішення, протоколи мітингів.

✓ Навіть при зміні членів команди знання не губляться.

3. Швидкий фідбек від користувачів

✓ UX-дизайнер проводить юзабіліті-тести й одразу показує результати команді.

✓ Продукт швидко адаптується під реальні потреби.

Погані приклади

1. Ігнорування стейкхолдерів

✓ Команда 3 місяці розробляє фічу без проміжних демо.

✓ Замовник бачить результат і каже: «Це не те, що ми хотіли».

2. Непрозора відповідальність

✓ Ніхто не знає, хто має інформувати інвесторів.

✓ Результат: інвестори дізнаються про затримку з чуток.

3. Надмірна кількість каналів

✓ Одночасно використовують Slack, Teams, Telegram і Email.

✓ Інформація дублюється, губиться, ніхто не знає, де актуальна версія.

Висновок:

✓ Хороша комунікація = регулярність + прозорість + релевантність.

✓ Погана комунікація = хаос + відсутність зворотного зв'язку + плутанина в каналах.

PM і BA повинні проектувати комунікаційні процеси так само уважно, як і технічні.

5.5. Конфлікти та методи їх вирішення

Конфлікт у команді – це зіткнення різних інтересів, цілей або підходів до виконання завдань.

5.5.1. Чому конфлікти виникають у командах

1. Природа конфлікту

Конфлікт не завжди є негативним: конструктивні конфлікти можуть

стимулювати інновації, тоді як деструктивні руйнують довіру й ефективність.

2. Основні причини конфліктів в ІТ-командах

1. Розподіл ресурсів

✓ Бюджет, час, доступ до інструментів чи обладнання.

✓ Приклад: QA вимагає більше часу на тестування, а Dev наполягають на швидкому релізі.

2. Різні очікування та пріоритети

✓ Бізнес хоче швидкий результат, команда орієнтується на якість.

✓ Приклад: Product Owner вимагає скорочення термінів, але архітектор попереджає про ризики.

3. Проблеми комунікації

✓ Непорозуміння через нечіткі вимоги, відсутність прозорої документації.

✓ Приклад: BA не зафіксував вимоги в Confluence, і розробники реалізували функціонал «по-своєму».

4. Різні стилі роботи й особистісні конфлікти

✓ Один член команди працює швидко, інший – методично і повільно.

✓ Приклад: конфлікт між «сеньйором», який хоче глибокої оптимізації коду, і «джуніором», який прагне швидко закрити задачу.

5. Зміни у проєкті

✓ Часта зміна вимог або стратегічних цілей.

✓ Приклад: замовник змінює пріоритети після вже зробленої роботи.

Типи конфліктів у команді

✓ Функціональні (конструктивні): допомагають знайти кращі рішення (дискусії про архітектуру, UX-дизайн).

✓ Деструктивні: шкодять відносинам і знижують ефективність (особистісні образи, саботаж).

Висновок. Конфлікти – це природна частина командної роботи, особливо в ІТ, де перетинаються різні ролі, інтереси й технічні підходи. Завдання PM та BA – вчасно виявляти джерела напруги, розрізняти конструктивні й деструктивні конфлікти та створювати умови для їхнього вирішення (рис.5.19).

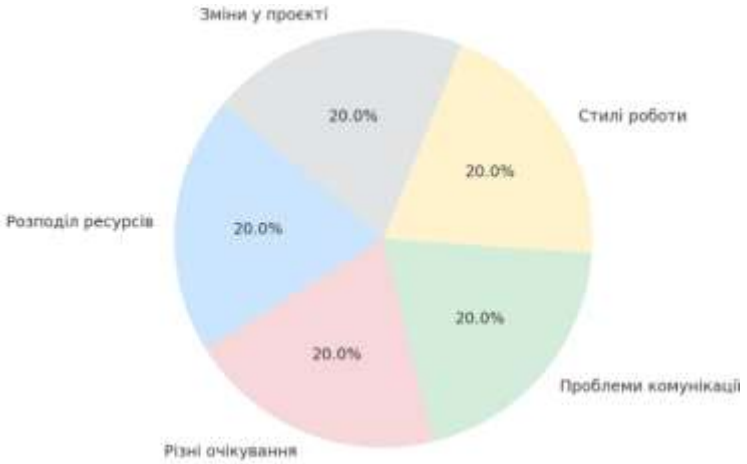

Рисунок 5.19 – Причини конфліктів у командах (приклад)

5.5.2. Типи конфліктів у командах: завдання, процеси, особистісні

1. Конфлікти завдань (Task Conflicts)

Визначення: виникають через різне бачення того, що саме потрібно робити і який результат вважається правильним.

Характеристика:

✓ стосуються змісту роботи, вимог до продукту чи пріоритетів;

✓ можуть бути конструктивними, якщо ведуть до уточнення вимог;

✓ стають деструктивними, якщо блокують прийняття рішень.

Приклад в IT:

✓ BA зафіксував вимогу «швидка автентифікація», але Dev і QA по-різному її трактують:

✓ Dev пропонує OAuth через Google/Facebook.

✓ QA вимагає SMS-підтвердження для безпеки.

Конфлікт вирішується через уточнення бізнес-цілей замовника.

2. Конфлікти процесів (Process Conflicts)

Визначення: виникають через розбіжності у тому, як саме потрібно виконувати роботу (процеси, ролі, ресурси).

-Характеристика:

✓ зачіпають правила взаємодії в команді;

✓ пов'язані з організацією робочого процесу, методологією (Scrum, Kanban, Waterfall), відповідальністю;

✓ часто призводять до затримок і неефективності.

Приклад в ІТ:

✓ DevOps наполягає на використанні CI/CD для швидких релізів.

✓ PM вимагає додаткових погоджень перед кожним релізом.

✓ Виникає суперечка: швидкість проти контролю.

Вирішення: узгодження Definition of Ready/Done та опис процесів у Confluence.

3. Особистісні конфлікти (Relationship Conflicts)

Визначення: конфлікти, що виникають через несумісність характерів, стилів спілкування чи особистих амбіцій.

Характеристика:

✓ найнебезпечніші, бо рідко стосуються роботи безпосередньо;

✓ впливають на атмосферу в команді, довіру та мотивацію;

✓ часто переростають у токсичність і демотивацію.

Приклад в ІТ:

✓ Senior-розробник постійно критикує джуніора на daily.

✓ Джуніор починає уникати ініціативи, продуктивність падає.

PM втручається, проводить фасилітацію, встановлює правила конструктивного фідбеку.

4. Взаємозв'язок типів конфліктів

Конфлікт завдань може легко перерости у конфлікт процесів (якщо сторони не погоджуються, як реалізувати завдання). Будь-який процесний конфлікт ризикує стати особистісним, якщо сторони не контролюють емоції. Завдання PM та BA – не допустити ескалації й перевести суперечку у конструктивне русло (рис.5.19).

Висновки. Не всі конфлікти є шкідливими: конфлікти завдань часто стимулюють креативність. Процесні конфлікти свідчать про необхідність чіткішої організації роботи. Особистісні конфлікти – найбільш руйнівні, потребують швидкого втручання. Ключ: своєчасне виявлення → аналіз → правильний метод вирішення (табл.5.6).

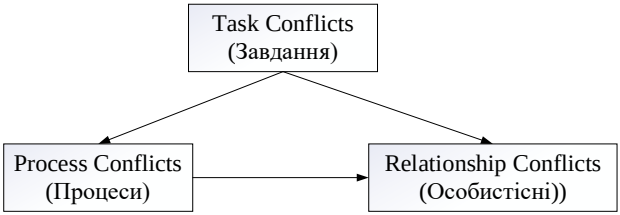

Рисунок 5.19 – Взаємозв'язок типів конфліктів

Таблиця 5.6 – Тип конфлікту → Приклад → Потенційний результат

Тип конфлікту	Приклад	Потенційний результат
Завдань	QA хоче SMS-підтвердження, Dev – OAuth	Уточнення вимог або блокування релізу
Процесів	DevOps за CI/CD, PM за додаткові погодження	Оптимізація процесу або затримка
Особистісні	Senior критикує джуніора на daily	Демотивація, токсичність, падіння продуктивності

5.5.3. Методи вирішення конфліктів (Thomas-Kilmann Conflict Mode Instrument, TKI)

Модель пропонує 5 основних стилів поведінки у конфлікті. Вони залежать від рівня асертивності (наполегливості у захисті власних інтересів) та кооперативності (готовності враховувати інтереси іншої сторони).

1. Уникання (Avoiding)

Суть: людина уникає обговорення проблеми, не захищає свої інтереси і не враховує інтереси іншої сторони.

Коли застосовується:

- ✓ конфлікт дріб'язковий;
- ✓ немає ресурсів чи часу;
- ✓ емоції надто сильні, і краще відкласти розмову.

Ризики: проблема залишається невирішеною, може загостритися з часом.

Приклад в IT: Dev і QA сперечаються про дрібний баг; PM вирішує відкласти обговорення до ретроспективи.

2. Пристосування (Accommodating)

Суть: людина жертвує власними інтересами заради іншої сторони.

Коли застосовується:

- ✓ важливіше зберегти стосунки, ніж відстояти свою позицію;
- ✓ питання не принципове;
- ✓ інша сторона має більше компетенції у питанні.

Ризики: постійне пристосування веде до втрати авторитету чи вигорання.

Приклад в ІТ: дизайнер погоджується на колірну схему, яку просить замовник, хоча вважає її менш вдалою.

3. Компроміс (Compromising)

Суть: кожна сторона частково відмовляється від своїх вимог, щоб дійти згоди.

Коли застосовується:

- ✓ потрібне швидке рішення;
- ✓ обидві сторони мають рівні позиції;
- ✓ неможливо задовольнити всі інтереси.

Ризики: рішення може бути середнім, але не оптимальним.

Приклад в ІТ: Dev хоче 3 тижні на фічу, замовник – 1 тиждень → домовляються про 2 тижні.

4. Співпраця (Collaborating)

Суть: пошук win-win рішення, яке максимально задовольняє обидві сторони.

Коли застосовується:

- ✓ проблема стратегічно важлива;
- ✓ є час і ресурси для дискусії;
- ✓ відносини між сторонами цінні й довгострокові.

Ризики: вимагає багато часу та довіри.

Приклад в ІТ: QA хоче більше тестів, Dev хоче швидкий реліз → команда впроваджує автоматизоване тестування.

5. Конкуренція (Competing)

Суть: одна сторона наполягає на своєму, не враховуючи інтереси іншої.

Коли застосовується:

- ✓ критична ситуація;
- ✓ потрібно швидке рішення;
- ✓ є чіткий авторитет (PM, CTO).

Ризики: може призвести до незадоволення й опору в команді.

Приклад в IT: реліз повинен відбутися до дедлайну → PM наказує деплоїти навіть з мінімальними тестами.

Висновок. У різних ситуаціях доречні різні методи (рис.5.20).

- ✓ Уникання та пристосування → для дрібних чи другорядних питань.
- ✓ Компроміс → для швидких угод.
- ✓ Співпраця → для стратегічних завдань.
- ✓ Конкуренція → для кризових випадків.

Рисунок 5.20 – Матриця Томаса-Кілмана з осями Асертивність-Кооперативність і 5 зонами (avoid, accommodate, compromise, collaborate, compete).

5.5.4. Роль лідера у вирішенні конфліктів

1. Чому лідерство важливе?

У більшості випадків конфлікти самостійно не зникають. Лідер (PM, Tech Lead або BA у певних випадках) – той, хто здатний помітити ознаки конфлікту на ранньому етапі й направити дискусію у конструктивне русло. Від дій лідера залежить, чи стане конфлікт джерелом розвитку, чи призведе до кризи в команді.

2. Основні ролі лідера в конфліктних ситуаціях:

Фасилітатор

✓ Допомагає сторонам почути одна одну.

✓ Створює безпечний простір для обговорення.

✓ Приклад: PM організовує медіацію між Dev і QA, які не можуть домовитися щодо пріоритетів багів.

Посередник (медіатор)

✓ Виступає «третьою стороною», що шукає компроміс.

✓ Пояснює різні точки зору й допомагає сформувати спільне рішення.

Арбітр

✓ У випадках, коли час критично обмежений, приймає остаточне рішення.

✓ Приклад: CTO вирішує питання архітектури, щоб команда не затягувала спринт.

Мотиватор

✓ Допомагає команді зрозуміти користь від вирішення конфлікту.

✓ Показує, як дискусії роблять продукт якіснішим.

Приклад для наслідування

✓ Демонструє конструктивну поведінку: вміння слухати, поважати чужу думку, не переходити на особистості.

3. Інструменти, якими користується лідер:

✓ Фасилітаційні техніки (наприклад, «правила дискусії», timeboxing).

✓ Методи активного слухання (перефразування, уточнення).

✓ Регулярні ретроспективи як профілактика.

✓ Розподіл відповідальності (RACI-матриця).

4. Важливі якості лідера при вирішенні конфліктів

✓ Емпатія.

✓ Неупередженість.

✓ Вміння працювати з емоціями.

✓ Готовність приймати непопулярні рішення.

Висновок. Лідер не завжди той, хто приймає рішення, але завжди той, хто задає тон спілкування та напрямок розвитку ситуації. Успішний PM чи BA вміє балансувати між ролями фасилітатора, посередника й арбітра (рис.5.21). Завдяки цьому конфлікти стають джерелом зростання команди, а не її руйнування.

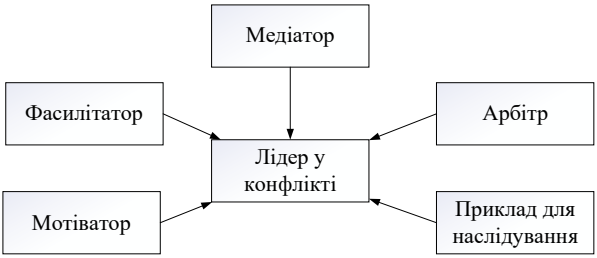

Рисунок 5.21 – Роль лідера у конфлікті:

5.5.5. Приклади вирішення конфліктів у IT-командах

1. Конфлікт завдань.

Ситуація: У команді стартапу Dev-розробники та UX-дизайнер мали різне бачення функціоналу реєстрації. Dev хотіли зробити мінімальну форму (email + пароль), а Дизайнер наполягав на соціальних логінах для зручності.

Дії лідера: PM організував короткий воркшоп: BA зібрав аргументи від обох сторін, а потім перевірив очікування замовника.

Рішення: В MVP додали мінімальну форму, а соцлогіни винесли в наступний спринт.

Результат: Задоволені обидві сторони, збережено швидкість релізу.

2. Конфлікт процесів

Ситуація: У великій аутсорсинговій компанії DevOps наполягав на автоматичному деплої через CI/CD, а QA-блок хотів залишити ручне підтвердження перед кожним релізом.

Дії лідера: CTO виступив як медіатор і проаналізував ризики.

Рішення: домовились:

- ✓ у staging-оточенні релізи автоматичні,
- ✓ у production – потрібне підтвердження QA.

Результат: збережено контроль і пришвидшено цикл розробки.

3. Особистісний конфлікт

Ситуація: Senior-розробник регулярно критикував джуніора, що призводило до його демотивації. На daily мітингах атмосфера ставала токсичною.

Дії лідера: PM провів індивідуальні бесіди та фасилітаційну сесію. Senior отримав фідбек про стиль комунікації, а для джуніора запровадили менторство.

Рішення: домовились, що критика буде подаватися лише у форматі code-review з конструктивними порадами.

Результат: атмосфера в команді покращилася, джуніор почав активніше брати участь у задачах.

4. Конфлікт з клієнтом (stakeholder conflict)

Ситуація: Замовник вимагав нову функцію напередодні релізу, що могло зірвати дедлайн.

Дії лідера: BA та PM підготували аналіз: вартість, ризики, вплив на терміни.

Рішення: домовились відкласти функцію на наступний реліз, натомість замовнику пообіцяли швидкий hotfix після старту.

Результат: реліз відбувся вчасно, клієнт отримав прозору комунікацію.

Сводна інформація по прикладам конфліктів наведена у таблиці.5.7.

Таблиця 5.7 – Приклади вирішення конфліктів у IT-командах

Тип	Ситуація	Дії лідера	Рішення	Результат
Конфлікт завдань	Dev хотіли мінімальну форму реєстрації, UX-дизайнер – соцлогіни.	PM організував воркшоп, BA уточнив очікування замовника.	MVP із простою формою, соцлогіни – у наступному спринті.	Реліз вчасно, обидві сторони задоволені.
Конфлікт процесів	DevOps за CI/CD, QA хотіли ручне підтвердження.	CTO виступив як медіатор, оцінив ризики.	Staging – автоматично, Production – із підтвердженням QA.	Цикл швидший, контроль збережено.
Конфлікт осіб	Senior критикував джуніора на daily, атмосфера ставала токсичною.	PM провів бесіди й фасилітацію, запровадив менторство.	Критика лише через code-review у конструктивному форматі.	Атмосфера покращилась, джуніор активніший.
Конфлікт з клієнтом	Замовник вимагав нову функцію напередодні релізу.	BA і PM підготували аналіз ризиків і термінів.	Функцію перенесли на наступний реліз, пообіцяли hotfix.	Реліз вчасно, клієнт задоволений прозорістю.

Висновки. Конфлікти у IT-командах різноманітні: від технічних дискусій до особистісних проблем. Ключ до вирішення – роль лідера як фасилітатора, медіатора чи арбітра. Приклади показують: навіть складні ситуації можуть перетворитися на можливість для покращення командної динаміки та продукту.

5.6. Загальні висновки

5.6.1. Основні тези лекції

1. Успіх IT-проєкту напряму залежить від командної динаміки та ефективності комунікацій.
2. Ролі в команді (Dev, QA, BA, PM, UX/UI, DevOps, Product Owner) формують баланс компетенцій і відповідальності.
3. Модель Такмана допомагає зрозуміти розвиток команди та роль лідера на кожному етапі.
4. Ефективна комунікація зі стейкхолдерами потребує вибору правильних каналів, інструментів та плану взаємодії.
5. Конфлікти неминучі, але їх можна перетворити на ресурс для зростання команди, якщо застосовувати методи Томаса-Кілмана й активну роль лідера.
6. Приклади з IT-практики показують, що навіть складні суперечки можуть призвести до зміцнення команди.

5.6.2. Питання для самоперевірки

1. Які основні ролі в IT-команді та їхні завдання?
2. У чому полягає модель розвитку команди за Такманом?
3. Хто такі стейкхолдери та чому вони критично важливі для проєкту?
4. Які інструменти комунікації використовуються в сучасних IT-командах?
5. У чому різниця між «хорошою» та «поганою» комунікацією?
6. Чому виникають конфлікти у командах?
7. Які типи конфліктів бувають (завдання, процеси, особистісні)?
8. Опиши 5 методів вирішення конфліктів за Томасом-Кілманом.

9. Яку роль відіграє лідер у вирішенні конфліктів?
10. Наведи приклад з практики, коли конфлікт став поштовхом до розвитку команди.

5.6.3. Завдання для самостійної роботи

✓ Побудувати комунікаційний план для вигаданого проєкту (наприклад, створення мобільного застосунку для навчання).

✓ Проаналізувати й змоделювати сценарій конфлікту між QA і Dev у спринті та запропонувати метод його вирішення.

✓ Визначити, на якій стадії за моделлю Такмана знаходиться ваша навчальна або робоча група.

5.7. Контрольні запитання

1. Чому командна робота є ключовим фактором успіху ІТ-проєкту? Наведіть аргументи та приклади.
2. Що таке синергія у командній роботі? Як вона проявляється в ІТ-команді?
3. Опишіть взаємозалежність ролей у типовій ІТ-команді (BA, розробник, тестувальник, PM).
4. Що таке стадії розвитку команди за моделлю Такмана (Forming, Storming, Norming, Performing)? Охарактеризуйте кожну.
5. Чому ефективна комунікація є основою управління ІТ-проєктом? Наведіть статистичні дані та обґрунтуйте.
6. Яким чином затримка або спотворення інформації впливає на строки та якість проєкту? Наведіть приклад.
7. Які інструменти і канали комунікації використовуються в сучасних ІТ-командах? Порівняйте їх переваги і недоліки.
8. Що таке план комунікацій у проєкті? Які елементи він повинен містити?
9. Як ефективна комунікація знижує ризики виникнення конфліктів у команді?
10. Що таке управління конфліктами в ІТ-проєкті? Назвіть основні стратегії вирішення конфліктів.

11. Як мотивація членів команди впливає на результати проєкту? Які теорії мотивації застосовуються у проєктному менеджменті?
12. Що таке матриця відповідальності (RACI)? Поясніть кожну роль і наведіть приклад для IT-проєкту.
13. Як проводяться ефективні наради (мітинги) в IT-проєктах? Які правила підвищують їх продуктивність?
14. Як управляти розподіленою командою (remote team)? Назвіть ключові виклики та способи їх вирішення.
15. Яку роль відіграє звітність і прозорість у комунікаціях між командою та стейкхолдерами?

6. УПРАВЛІННЯ РЕСУРСАМИ ТА РИЗИКАМИ

6.1. Вступ

Значення ефективного управління ресурсами та ризиками в ІТ-проєктах

1. Чому це важливо?

Успіх будь-якого ІТ-проєкту залежить не тільки від коду чи технологій, а й від того, як правильно розподілені ресурси та як команда справляється з ризиками. Погане управління ресурсами може призвести до вигорання команди, перевищення бюджету або зриву термінів. Ігнорування ризиків часто стає причиною провалів навіть у великих компаніях.

2. Приклади з практики

✓ Overbooking ресурсів: компанія одночасно запустила кілька проєктів, використовуючи тих самих розробників. У результаті – зрив дедлайнів у всіх проєктах.

✓ Непередбачені ризики: у 2020 році багато ІТ-компаній були змушені терміново переходити на віддалену роботу. Ті, хто мав ризик-плани (наприклад, інструменти для онлайн-комунікації та резервні сервери), адаптувалися швидко. Інші втратили клієнтів і прибутки.

✓ Класичний кейс: проєкт Denver Airport Baggage System (США, 1990-ті) – через неправильне управління ресурсами й відсутність аналізу ризиків система не запрацювала вчасно, що коштувало місту сотні мільйонів доларів.

3. Взаємозв'язок ресурсів і ризиків

✓ Ресурси: люди, час, гроші, обладнання, інформація.

✓ Ризики: події, які можуть негативно вплинути на використання ресурсів.

✓ Якщо ресурси сплановані неправильно → ризик зростає.

✓ Якщо ризики не враховані → навіть достатні ресурси можуть бути витрачені дарма.

Висновок. Управління ресурсами та ризиками – це серце проєктного менеджменту (рис.6.1). Для ІТ-проєктів, де невизначеності завжди багато (зміни вимог, технічні обмеження, людський фактор), цей блок управління є

критичним. PM та BA повинні вміти балансувати ресурси й прогнозувати ризики, щоб команда працювала стабільно та ефективно.

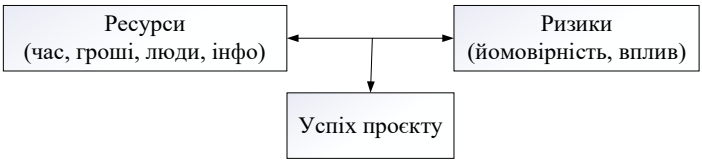

Рисунок 6.1 – Взаємозв'язок ресурсів і ризиків

Приклади провалів через погане планування в IT-проєктах

1. Недостатній бюджет

Кейс: стартап планував створити маркетплейс із повною інтеграцією платежів і логістики, але бюджет був розрахований тільки на MVP.

Наслідки: команда витратила гроші на часткову реалізацію і не змогла залучити інвестиції, бо продукт виглядав «незавершеним».

Висновок: відсутність чіткого планування бюджету та пріоритетів призводить до провалу.

2. Вигорання команди

Кейс: у середній IT-компанії PM неправильно розрахував обсяг робіт і навантажив команду понад норму.

Наслідки: розробники працювали понаднормово 3 місяці, у результаті кілька ключових спеціалістів звільнилися, а швидкість розробки впала ще більше.

Висновок: перевантаження ресурсів у короткостроковій перспективі «вбиває» команду в довгостроковій.

3. Технічні збої через відсутність резервів

Кейс: у банківському проєкті не було передбачено резервних серверів. При зростанні кількості користувачів система «лягла».

Наслідки: кілька днів клієнти не могли користуватися онлайн-банкінгом, компанія втратила репутацію і значні кошти.

Висновок: технічні ресурси треба планувати з урахуванням масштабування та резервів.

4. Класичні приклади з історії

✓ Denver Airport Baggage System (США, 1990-ті): через відсутність планування ризиків і недооцінку складності система автоматизації багажу не запрацювала вчасно. Втрати – понад $500 млн.

✓ NHS National Programme for IT (Велика Британія, 2000-ті): один з найбільших провалів у державному IT – проєкт вартістю £10 млрд закрили через перевитрати та проблеми з управлінням ресурсами.

Висновок. Більшість провалів у IT-проєктах трапляються не через технології, а через помилки у плануванні ресурсів і ризиків (табл.А.1). Це підкреслює необхідність навчати майбутніх менеджерів системному мисленню й роботі з прогнозуванням.

6.2. Планування та розподіл ресурсів

Управління ресурсами – фундамент ефективності проєкту.

6.2.1. Види ресурсів

1. Людські ресурси

Це найцінніший актив будь-якого IT-проєкту. Включають: розробників, тестувальників, бізнес-аналітиків, дизайнерів, DevOps, менеджерів.

Особливість: унікальні компетенції та різний рівень досвіду.

Виклики:

✓ Перевантаження або недовантаження спеціалістів.

✓ Втрата ключового експерта.

✓ Необхідність мотивації та підтримки командного духу.

Інструменти: матриця RACI, планування спринтів, capacity planning.

2. Фінансові ресурси

Гроші, виділені на проєкт (бюджет). Включають: зарплати команди, інфраструктуру (сервери, ліцензії, сервіси), маркетинг.

Виклики:

✓ Недооцінка реальної вартості робіт.

✓ Перевитрати через зміни вимог.

✓ Ризик «замороження» фінансування.

Інструменти: cost baseline, burn rate, планування бюджету по фазах.

3. Матеріальні ресурси

У класичних проєктах – офіс, меблі, техніка. В IT – це також робочі місця, ноутбуки, тестові пристрої (смартфони, планшети), серверні приміщення.

Виклики:

- ✓ Застаріле обладнання.
- ✓ Несправність техніки.
- ✓ Витрати на обслуговування.

Інструменти: інвентаризація, план оновлення обладнання.

4. Технічні ресурси

Інфраструктура, необхідна для розробки:

- ✓ сервери, хмарні сервіси (AWS, Azure, GCP),
- ✓ CI/CD-системи,
- ✓ інструменти моніторингу,
- ✓ середовища тестування.

Виклики:

- ✓ Масштабування при зростанні користувачів.
- ✓ Безпека даних.
- ✓ Сумісність технологій.

Інструменти: хмарні сервіси, DevOps-практики, резервні системи.

Висновок. Для успішного проєкту потрібно балансувати чотири типи ресурсів: людські, фінансові, матеріальні та технічні (рис.6.2). Надмірна увага до одного типу (наприклад, фінансів) без урахування інших (людей чи техніки) майже завжди призводить до ризиків і провалів.

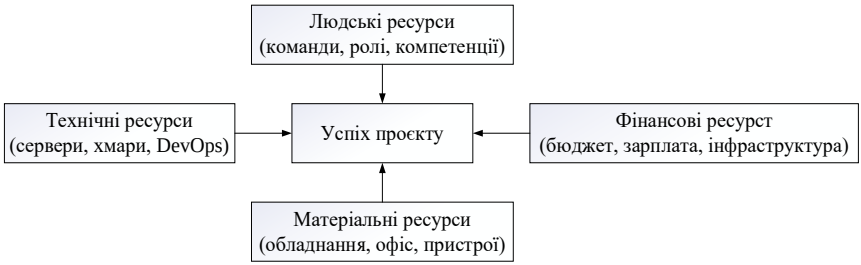

Рисунок 6.2 – Види ресурсів і їхній вплив на успіх проєкту

6.2.2. Методи оцінки потреб у ресурсах

1. Expert Judgment (експертна оцінка)

Найпоширеніший метод. Передбачає залучення досвідчених спеціалістів (PM, технічних лідів, архітекторів) для оцінки потреб у людських, фінансових і технічних ресурсах. Використовується на ранніх етапах, коли ще немає повної інформації.

Приклад: техлід оцінює, що для створення модулю потрібні 2 розробники протягом 3 місяців.

2. Аналогове оцінювання

Базується на порівнянні з минулими проєктами. Якщо раніше створення мобільного застосунку подібного масштабу зайняло 6 місяців і команду з 5 осіб, то новий проєкт оцінюється приблизно так само.

Мінус: точність залежить від схожості проєктів.

3. Параметричне оцінювання

Використовує математичні моделі. Наприклад: якщо відомо, що розробка однієї функції в середньому займає 20 годин, а у проєкті 50 функцій → потрібно 1000 годин.

Плюс: підвищує об’єктивність оцінки.

Мінус: потребує статистичних даних.

4. Bottom-Up Estimation (оцінка «знизу-вгору»)

Завдання декомпонується на дрібніші частини. Для кожної частини визначаються ресурси, а потім результати сумуються. Найточніший метод, але потребує багато часу.

Приклад: створення вебсайту оцінюється окремо по фронтенду, бекенду, дизайну, тестуванню.

5. Planning Poker (Agile-проєкти)

Використовується в Scrum і Kanban. Команда спільно оцінює складність завдань (наприклад, у story points), після чого плануються ресурси.

Перевага: враховує думку всієї команди.

Недолік: суб’єктивність.

Висновок. Жоден метод не є універсальним (табл.А.2). У практиці проєктного менеджменту часто поєднують кілька підходів:

- ✓ на ранніх стадіях – експертні оцінки та аналоги,
- ✓ під час деталізації – bottom-up та параметричні методи,
- ✓ в Agile-командах – Planning Poker.

6.2.3. Баланс між обсягом робіт, часом і вартістю

1. Суть «трикутника управління проєктом»

Відомий також як Project Management Triangle або Iron Triangle. Складається з трьох ключових параметрів:

- ✓ Обсяг робіт (Scope) – що саме потрібно зробити, які функції й вимоги реалізувати.
- ✓ Час (Time) – скільки є часу на виконання проєкту.
- ✓ Вартість (Cost) – який бюджет доступний для реалізації.

2. Принцип взаємозалежності

Змінюючи одну сторону трикутника, ми впливаємо на дві інші.

Приклади:

- ✓ Якщо замовник хоче більше функціоналу (зростає Scope) → треба більше часу або грошей.
- ✓ Якщо скорочують терміни (Time) → зростає бюджет (Cost) або зменшується функціонал.
- ✓ Якщо урізають бюджет (Cost) → доведеться скоротити Scope або подовжити Time.

3. Класичне правило

- ✓ Неможливо одночасно зробити швидко, дешево й якісно.
- ✓ Команда й замовник завжди мають робити компроміси.

4. Практичний приклад

Проєкт зі створення мобільного застосунку для доставки їжі.

- ✓ Scope: користувацька частина, адмін-панель, інтеграція з платіжними системами.
- ✓ Time: замовник вимагає реліз за 3 місяці.
- ✓ Cost: бюджет обмежений.

Вихід: скоротити Scope (залишити тільки базові функції) → випустити MVP, а розширений функціонал доробити у наступних версіях.

Висновок. Трикутник управління – це інструмент для пояснення

замовникам і стейкхолдерам, чому неможливо змінювати один параметр без впливу на інші (рис.6.2). Його знання допомагає уникати нереалістичних очікувань.

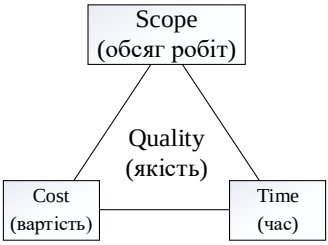

Рисунок 6.3 – Трикутник управління проєктом

6.2.4. Інструменти для планування та розподілу ресурсів

1. MS Project

Класичний інструмент для планування проєктів.

Підтримує: діаграми Ґанта, визначення критичного шляху, управління бюджетом і завантаженням ресурсів.

Сильні сторони:

- ✓ глибока деталізація,
- ✓ інтеграція з Microsoft 365,
- ✓ підходить для великих і комплексних проєктів.

Виклик: складний у навчанні, потребує ліцензії.

2. Jira

Найпопулярніший інструмент в ІТ-командах, особливо в Agile. Можна відслідковувати завантаження команди та ефективність використання ресурсів.

Підтримує: backlog, спринти, kanban- і scrum-дошки.

Сильні сторони:

- ✓ гнучкість,
- ✓ інтеграції (Confluence, Bitbucket, Slack),
- ✓ автоматизація процесів.

Виклик: може бути «перевантаженою» для маленьких команд.

3. Trello

Простий і візуальний інструмент для управління завданнями. Добре підходить для невеликих команд або стартапів.

Використовує kanban-дошки, списки й картки.

Сильні сторони:

- ✓ інтуїтивний інтерфейс,
- ✓ швидке впровадження,
- ✓ безкоштовна версія.

Виклик: обмежені можливості для масштабних проєктів.

4. Resource Guru

Спеціалізований інструмент саме для управління ресурсами.

Дозволяє: планувати завантаження співробітників, керувати відпустками, балансувати workload.

Сильні сторони:

- ✓ проста візуалізація розподілу людей,
- ✓ календар доступності,
- ✓ інтеграція з Google Calendar, Outlook.

Виклик: немає повного функціоналу управління проєктами (тільки ресурси).

Висновок (табл.А.3):

- ✓ Для великих корпоративних проєктів підходить MS Project.
- ✓ Для гнучких команд і Agile – Jira.
- ✓ Для невеликих стартапів – Trello.
- ✓ Для управління саме людськими ресурсами – Resource Guru.

6.3. Інструменти управління часом

6.3.1. Діаграма Ґанта

1. Принцип

Діаграма Ґанта – це графік у вигляді горизонтальних смуг, що показує:

- ✓ завдання проєкту,
- ✓ їхню тривалість,
- ✓ послідовність виконання,
- ✓ залежності між завданнями.

Вперше запропонована Генрі Ґантом у 1910-х роках.

Використовується для візуального відображення календарного плану проєкту.

2. Приклад

Завдання:

- ✓ Дизайн інтерфейсу (2 тижні),
- ✓ Розробка бекенду (4 тижні),
- ✓ Тестування (2 тижні).

У діаграмі видно, що дизайн починається першим, розробка йде після нього, а тестування – наприкінці (рис.6.3).

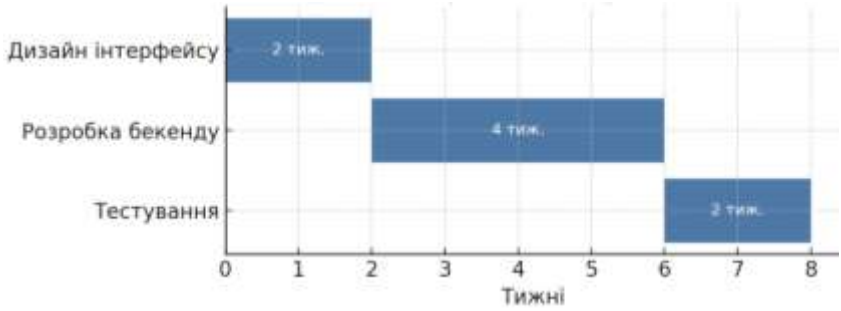

Рисунок 6.3 – Приклад простої діаграми Ґанта

3. Переваги:

- ✓ Простота і наочність – легко пояснити замовнику чи стейкхолдерам.
- ✓ Дає змогу бачити загальну картину проєкту.
- ✓ Дозволяє визначати критичні завдання і залежності.
- ✓ Корисна для моніторингу прогресу.

4. Недоліки:

- ✓ Складно підтримувати актуальність у великих і динамічних проєктах.
- ✓ При сотнях завдань діаграма стає громіздкою і незручною.
- ✓ Погано підходить для Agile, де плани часто змінюються.
- ✓ Вимагає ручного оновлення (у простих інструментах).

Висновок. Діаграма Ґанта залишається класичним і дуже корисним інструментом для традиційних та змішаних методологій. В IT-проєктах її часто застосовують для верхньорівневого планування (roadmap), а в деталях переходять до Agile-дошок.

6.3.2. Метод критичного шляху (CPM – Critical Path Method)

1. Суть методу.

CPM – це спосіб знайти найдовший шлях у мережі завдань проєкту, тобто послідовність робіт, яка визначає мінімальний час виконання всього проєкту. Якщо завдання на критичному шляху затримується – затримується весь проєкт.

2. Як працює.

✓ Визначити завдання і їхню тривалість.

✓ Позначити залежності (які завдання повинні завершитися, щоб почалися інші).

✓ Побудувати мережеву діаграму.

✓ Розрахувати тривалість усіх можливих шляхів.

✓ Найдовший шлях = критичний.

3. Приклад.

Завдання (рис.6.4):

✓ A: Аналіз вимог (2 дні).

✓ B: Дизайн (3 дні, після A).

✓ C: Розробка (5 днів, після B).

✓ D: Тестування (2 дні, після C).

✓ E: Документація (2 дні, може йти паралельно з D).

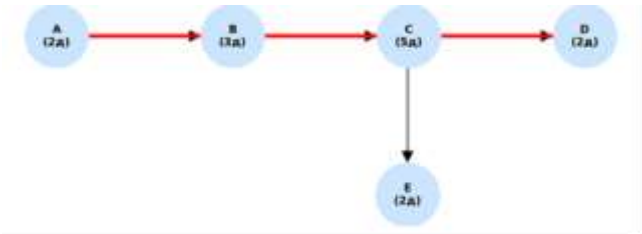

Рисунок 6.4 – Схема мережевої діаграми

Критичний шлях: A → B → C → D = 12 днів.

Навіть якщо документація (E) затримається, проєкт не зірветься, бо вона не на критичному шляху.

4. Переваги:

✓ Допомагає побачити «вузькі місця».

✓ Чітко показує мінімальний термін завершення проєкту.

✓ Дає змогу виділити завдання, на які потрібно звертати найбільшу увагу.

5. Недоліки

✓ Складність у великих проєктах із сотнями завдань.

✓ Не враховує ризиків і ресурсних обмежень сам по собі (часто комбінують із іншими методами).

Висновок. CPM – це базовий інструмент управління часом, який часто поєднують із діаграмами Ґанта. У реальних ІТ-проєктах він допомагає прогнозувати терміни та уникати зривів дедлайнів.

6.3.3. Метод критичного шляху (CPM): залежності та вузькі місця

1. Залежності між завданнями (табл.6.1)

Таблиця 6.1 – CPM для ІТ: залежності та вузькі місця

Тип залежності	Приклад у ІТ-проєкті
Finish-to-Start (FS)	Код пишеться після завершення дизайну.
Start-to-Start (SS)	Тестування може початися одночасно з частиною розробки.
Finish-to-Finish (FF)	Документація завершується одночасно з фінальним релізом.
Start-to-Finish (SF)	Стара система відключається тільки після запуску нової.

У проєктах завдання рідко виконуються ізольовано. Між ними виникають залежності:

✓ Finish-to-Start (FS) – наступне завдання починається тільки після завершення попереднього (найчастіше).

✓ Start-to-Start (SS) – завдання можуть стартувати одночасно, але одне залежить від іншого.

✓ Finish-to-Finish (FF) – обидва завдання повинні завершитися одночасно.

✓ Start-to-Finish (SF) – рідкісна залежність: нове завдання має початися, щоб інше завершилось.

У CPM будується мережна діаграма саме на основі цих залежностей.

2. Як виявляють «вузькі місця»?

Після побудови мережі завдань визначаємо усі можливі шляхи від початку до завершення.

Найдовший шлях = критичний шлях → він визначає мінімальну тривалість проєкту.

Завдання, що входять у критичний шлях, мають нульовий запас часу (float). Якщо хоча б одне із цих завдань затримується → затримується весь проєкт.

3. Приклад вузького місця

У розробці ПЗ:

Аналіз вимог (2 дні) → Дизайн (3 дні) → Розробка (5 днів) → Тестування (2 дні).

Критичний шлях = 12 днів.

Якщо тестування потребує більше часу (наприклад, 4 дні замість 2), то проєкт затримується на 2 дні.

Висновок. CPM – це система раннього попередження: вона показує, на які завдання варто звернути особливу увагу. Саме через пошук вузьких місць менеджери проєктів використовують CPM у поєднанні з діаграмами Ґанта.

6.3.4. Використання інструментів управління часом у сучасних IT-командах

1. Діаграма Ґанта у практиці.

Часто використовується для верхньорівневого планування (roadmap на кілька місяців). Дає змогу замовникам і стейкхолдерам бачити «картину зверху». У невеликих командах її будують у MS Project, Excel, Notion, ClickUp. У великих компаніях інтегрують із системами ресурсного планування.

2. Метод критичного шляху (CPM).

Використовується менеджерами для контролю термінів у складних проєктах. Наприклад: при міграції системи на нову платформу, де сотні залежностей. Дає змогу виділити ключові завдання, які не можна затримувати. У практиці часто інтегрований у MS Project, Primavera, Wrike.

3. Поєднання з Agile.

У класичному вигляді Gantt і CPM погано підходять для Agile. Але команди адаптують:

✓ Roadmap у вигляді діаграми Ґанта → для показу напрямку розвитку продукту.

✓ Критичні шляхи у спринтах → для визначення найважливіших залежностей.

Таким чином, інструменти класики працюють разом із Scrum- і Kanban-практиками.

4. Приклад із сучасної IT-команди.

Стартап із розробки мобільного додатку:

✓ Gantt – використали для планування релізу MVP протягом 3 місяців.

✓ CPM – застосували для визначення критичних завдань (бекенд і інтеграція з платіжними системами).

✓ Agile-дошка в Jira – використовувалась щодня для спринтів.

Результат: замовник бачив roadmap, а команда мала гнучкість у виконанні завдань.

Висновок. Сучасні IT-команди комбінують традиційні інструменти управління часом (Gantt, CPM) із Agile-підходами (рис.6.5). Це дозволяє поєднати прозорість для стейкхолдерів і гнучкість для розробників.

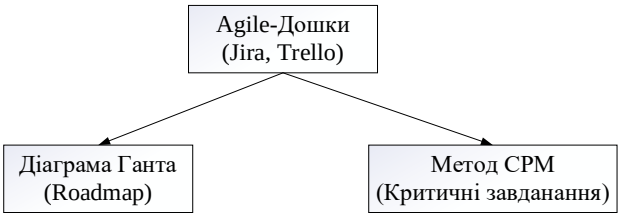

Рисунок 6.5 – Комбінація інструментів «Gantt + CPM + Agile» (прозорість+гнучкість).

6.4. Типи ризиків в ІТ-проєктах

Наявність ризиків не можливо уникнути, а зменшити їх вплив – задача, яка вирішується.

6.4.1. Технічні ризики

1. Суть технічних ризиків

Технічні ризики виникають тоді, коли проєкт стикається з проблемами, пов'язаними з технологіями, інфраструктурою чи якістю коду. Вони напряму впливають на якість продукту, швидкість розробки та можливість подальшої підтримки.

2. Приклади технічних ризиків

✓ Баги (помилки у коді): невиявлені дефекти на ранніх етапах можуть спричинити серйозні збої на продакшені.

✓ Несумісність технологій: використання бібліотек чи API, які не працюють разом або конфліктують.

✓ Застарілі технології: вибір фреймворку, який перестає підтримуватися, створює ризики для безпеки та майбутніх оновлень.

✓ Проблеми з продуктивністю: система не витримує навантаження (наприклад, 10 000 користувачів одночасно).

✓ Інтеграційні ризики: сторонні сервіси змінюють API, що ламає інтеграції.

3. Причини виникнення

✓ Погана оцінка технічних вимог.

✓ Недостатнє тестування.

✓ Використання «сирих» або експериментальних технологій.

✓ Відсутність архітектурного аналізу.

4. Наслідки

✓ Додаткові витрати на виправлення.

✓ Подовження термінів розробки.

✓ Зниження якості продукту та репутаційні втрати.

5. Як мінімізувати технічні ризики

✓ Використовувати Code Review та автоматичне тестування.

✓ Робити Proof of Concept (PoC) для нових технологій перед

масштабуванням.

✓ Залучати архітектора на ранніх етапах.

✓ Використовувати CI/CD для швидкого виявлення багів.

✓ Регулярно оновлювати залежності та стежити за підтримкою бібліотек.

Висновок. Технічні ризики є одними з найпоширеніших у ІТ-проєктах (табл.А.4). Їх можна зменшити завдяки системному тестуванню, ретельному вибору технологій та використанню сучасних практик CI/CD і DevOps.

6.4.2. Бізнес-ризики

1. Суть бізнес-ризиків

Бізнес-ризики пов’язані з нестабільністю бізнес-середовища, замовника або ринку, на якому розробляється продукт. Вони впливають на доцільність, фінансову ефективність та кінцеву цінність проєкту.

2. Приклади бізнес-ризиків

✓ Зміна вимог: замовник радикально переглядає бачення продукту, що призводить до переробок.

✓ Втрати клієнта: компанія може втратити ключового замовника чи користувача, для якого створювався продукт.

✓ Зміна ринку: вихід конкурентів із кращим продуктом робить проєкт менш актуальним.

✓ Зміна регуляцій: нові закони (наприклад, GDPR, податкові вимоги) потребують термінових змін у системі.

✓ Фінансові проблеми замовника: скорочення бюджету чи закриття бізнесу.

3. Причини виникнення

✓ Недостатня робота з вимогами на старті.

✓ Слабкий аналіз ринку.

✓ Відсутність регулярної комунікації із замовником.

✓ Залежність від одного клієнта.

4. Наслідки

✓ Зростання витрат на доопрацювання.

✓ Подовження термінів.

✓ Ризик зупинки або замороження проєкту.

✓ Втрата конкурентних переваг.

5. Як мінімізувати бізнес-ризики

✓ Використовувати Agile-підходи – часті релізи дозволяють швидше враховувати зміни.

✓ Документувати зміни вимог і застосовувати Change Management.

✓ Регулярна комунікація зі стейкхолдерами (демо, ретроспективи).

✓ Диверсифікація клієнтської бази – не залежати від одного замовника.

✓ Проведення аналізу ринку та конкурентів перед і під час проєкту.

Висновок. Бізнес-ризики можуть суттєво вплинути на успіх проєкту (таюлА.5). Їх можна зменшити завдяки Agile-підходам, ефективній комунікації зі стейкхолдерами та аналізу ринку й фінансового стану замовників.

6.4.3. Організаційні ризики

1. Суть організаційних ризиків

Організаційні ризики пов’язані з людськими ресурсами, структурою управління та процесами в компанії чи команді. Вони напряму впливають на ефективність роботи, швидкість виконання завдань і якість кінцевого продукту.

2. Приклади організаційних ризиків

✓ Висока плинність кадрів: ключові спеціалісти залишають проєкт, знання втрачаються.

✓ Конфлікти в команді: суперечки між розробниками, дизайнерами, тестувальниками.

✓ Неефективний менеджмент: погане планування, відсутність контролю.

✓ Низька кваліфікація команди: завдання виконуються довше або з помилками.

✓ Проблеми з комунікаціями: інформація не доходить до стейкхолдерів або перекручується.

3. Причини виникнення

✓ Відсутність чіткої структури командної роботи.

✓ Недостатня мотивація співробітників.

✓ Погано налагоджені комунікаційні канали.

✓ Перевантаження або вигорання.

4. Наслідки

✓ Затримки у виконанні завдань.

✓ Зниження продуктивності та якості.

✓ Втрата ключових спеціалістів.

✓ Ризик зриву термінів і контрактних зобов'язань.

5. Як мінімізувати організаційні ризики

✓ Формувати чіткі ролі та зони відповідальності в команді.

✓ Використовувати методології управління проєктами (Scrum, Kanban, PMBOK).

✓ Забезпечувати мотивацію та розвиток персоналу (курси, менторство, бонуси).

✓ Регулярні ретроспективи й робота з конфліктами.

✓ Використання інструментів для комунікацій (Slack, Jira, Confluence).

Висновок. Організаційні ризики часто стають ключовим фактором зриву проєктів (табл.А.6). Їх можна мінімізувати завдяки чіткому управлінню ролями, розвитку команди та налагодженим комунікаціям.

6.4.4. Зовнішні ризики

1. Суть зовнішніх ризиків

Зовнішні ризики виникають унаслідок факторів, які не залежать від команди чи компанії, але мають суттєвий вплив на проєкт. Це ризики з бізнес-середовища, політики, економіки, суспільства чи форс-мажорних обставин.

2. Приклади зовнішніх ризиків

✓ Зміни в законодавстві: нові вимоги до безпеки даних, ліцензування.

✓ Економічна нестабільність: інфляція, коливання валют, що впливають на бюджет.

✓ Політичні фактори: війна, санкції, обмеження експорту технологій.

✓ Форс-мажори: пандемії, природні катастрофи, відключення енергії.

✓ Ринкові ризики: падіння попиту на продукт через зміну трендів.

3. Причини виникнення

✓ Залежність від державних чи міжнародних регуляцій.

✓ Висока чутливість бізнесу до економічних і політичних процесів.

✓ Недостатній аналіз зовнішнього середовища.

4. Наслідки

✓ Призупинення або зупинка проєкту.

✓ Різке збільшення бюджету.

✓ Втрата ринку чи клієнтів.

✓ Неможливість виконати контрактні зобов'язання.

5. Як мінімізувати зовнішні ризики

✓ Використовувати аналіз PESTEL (політичні, економічні, соціальні, технологічні, екологічні, правові фактори).

✓ Розробляти плани безперервності бізнесу (Business Continuity Plan).

✓ Мати резервний бюджет і технічні рішення (backup сервери, альтернативні постачальники).

✓ Юридичний супровід і постійний моніторинг регуляцій.

✓ Страхування ризиків (у великих компаніях).

Висновок. Зовнішні ризики є неконтрольованими, але їхній вплив можна зменшити завдяки плануванню, резервам, аналізу PESTEL і планам безперервності бізнесу (табл.А.7).

6.4.5. Приклади реальних кейсів

1. Технічний ризик: помилка у фінансовій системі

Ситуація: У банківській системі некоректний алгоритм округлення призвів до неправильного розрахунку відсотків по депозитах.

Причина: Недостатнє тестування та пропуск критичного сценарію.

Наслідок: Банку довелося компенсувати клієнтам втрати, що коштувало мільйони доларів.

Урок: Навантажувальне й автоматизоване тестування повинно бути частиною процесу з першого дня.

2. Бізнес-ризик: зміна стратегії замовника

Ситуація: Стартап із розробки мобільного застосунку для подорожей втратив інвестора під час пандемії COVID-19.

Причина: Різке падіння попиту на подорожі.

Наслідок: Проєкт заморозили на 2 роки, команда була розпущена.

Урок: Потрібна стратегія «B-plan» і можливість швидко змінювати бізнес-модель.

3. Організаційний ризик: плинність кадрів у корпорації

Ситуація: У великій продуктовій компанії ключовий архітектор залишив команду посеред проєкту.

Причина: Відсутність плану передачі знань і низька мотивація.

Наслідок: Розробка затрималась на 6 місяців, оскільки нова команда довго розбиралась у коді.

Урок: Важливо мати knowledge transfer та документацію архітектури.

4. Зовнішній ризик: відключення електропостачання

Ситуація: Українська ІТ-компанія під час війни у 2022-2023 рр. регулярно стикалася з блекаутами.

Причина: Зовнішні обставини, які неможливо контролювати.

Наслідок: Робота команд зупинялась на кілька годин або днів.

Урок: Використання генераторів, UPS, коворкінгів з автономним живленням та віддалених команд.

Висновок. Реальні кейси демонструють, що ризики існують на всіх рівнях: технічному, бізнесовому, організаційному та зовнішньому (табл.А.8). Успішні компанії відрізняються тим, що не лише виявляють ризики заздалегідь, але й мають стратегії їх мінімізації.

6.5. Методи аналізу та мінімізації ризиків

Кількісна та якісна оцінки ризику дозволяю зменшити її вплив на результат проєкту, а для цього використовуються різноманітні засоби.

6.5.1. Risk Register (реєстр ризиків)

1. Що таке Risk Register?

Реєстр ризиків (Risk Register) – це документ або база даних, де системно збирається інформація про всі виявлені ризики у проєкті. Він є центральним

інструментом управління ризиками у PMBOK, PRINCE2 та інших методологіях.

2. Призначення реєстру ризиків

- ✓ Централізує всі дані про ризики.
- ✓ Дозволяє відстежувати статус кожного ризику.
- ✓ Забезпечує прозорість для стейкхолдерів.
- ✓ Допомагає приймати рішення про пріоритети.

3. Типові колонки у Risk Register (табл.6.2):

- ✓ ID ризику (унікальний номер).
- ✓ Опис ризику (що може статися).
- ✓ Категорія (технічний, бізнесовий, організаційний, зовнішній).
- ✓ Ймовірність виникнення (низька, середня, висока).
- ✓ Вплив (незначний, середній, критичний).
- ✓ Рівень ризику (комбінація ймовірності та впливу).
- ✓ Власник ризику (хто відповідає за моніторинг і реакцію).
- ✓ Стратегія реагування (уникання, зменшення, передача, прийняття).
- ✓ Поточний статус (активний, знижений, закритий).

Таблиця 6.2 – Приклад Risk Register (для IT-проєкту)

ID	Опис ризику	Категорія	Ймовірність	Вплив	Рівень	Власник	Стратегія	Статус
R1	Втрата ключового розробника	Організаційний	Середня	Високий	Високий	PM	Knowledge transfer, резерв	Активний
R2	Зміна законодавства GDPR	Зовнішній	Низька	Високий	Середній	Юрист	Моніторинг, адаптація системи	Активний
R3	Перевантаження серверів під час релізу	Технічний	Висока	Високий	Критичний	DevOps	Навантажувальне тестування	Закритий

Висновок. Реєстр ризиків – це «жива» база, яка оновлюється протягом усього життєвого циклу проєкту. Його правильне ведення допомагає знизити невизначеність і підвищує передбачуваність результатів.

6.5.2. Risk Matrix (матриця ймовірність × вплив)

1. Що таке Risk Matrix?

Матриця ризиків – це візуальний інструмент для оцінки рівня ризику залежно від (табл.6.3):

- ✓ Ймовірності виникнення (Likelihood)
- ✓ Впливу на проєкт (Impact)

Кожен ризик позначається у матриці, що дозволяє швидко визначити його пріоритет.

2. Типова структура матриці (рис.6.7).

Матриця має дві осі:

- ✓ Y (вертикальна): Ймовірність (низька, середня, висока).
- ✓ X (горизонтальна): Вплив (низький, середній, високий).

Клітинки кольорово позначають рівень небезпеки:

- ✓ Зелений → низький ризик (можна прийняти).
- ✓ Жовтий → середній ризик (потрібно контролювати).
- ✓ Червоний → високий ризик (необхідні дії).

Таблиця 6.3 – Приклад матриці ризиків (для ІТ-проєкту)

Ймовірність / Вплив	Низький	Середній	Високий
Висока	Середній	Високий	Критичний
Середня	Низький	Середній	Високий
Низька	Низький	Низький	Середній

Рисунок 6.7 – Схема матриці ризиків

4. Використання

- ✓ Допомагає швидко оцінити пріоритетність ризиків.
- ✓ Використовується у плануванні для вибору стратегії реагування.
- ✓ Легко пояснити замовнику чи керівництву рівень загрози.

Розширена матриця (4x4 чи 5x5) дозволяє більш гнучко оцінювати ризики. Вона надає проєктним менеджерам візуальний інструмент для пріоритизації дій та вибору стратегії реагування (таюл.5.4, рис.6.8).

Таблиця 6.12 – Прикладів ризиків

Ризик	Ймовірність	Вплив	Категорія у матриці
Втрата ключового розробника	Середня	Високий	Червона зона – критичний ризик
Зміна законодавства (GDPR)	Низька	Критичний	Помаранчева зона – значний ризик
Перевантаження серверів	Висока	Середній	Жовта зона – середній ризик
Затримка з боку клієнта	Середня	Низький	Зелена зона – допустимий ризик

Рисунок 6.8 – Приклад 4x4 матриці ризиків

Висновок. Risk Matrix – один із найзрозуміліших інструментів управління ризиками, який дозволяє поєднати якісну та кількісну оцінку.

6.5.3. SWOT-аналіз

1. Що таке SWOT-аналіз?

SWOT-аналіз – це метод стратегічного планування, який дозволяє оцінити:

S (Strengths) – Сильні сторони

W (Weaknesses) – Слабкі сторони

O (Opportunities) – Можливості

T (Threats) – Загрози

У контексті управління ризиками SWOT допомагає збалансувати внутрішні та зовнішні фактори, які можуть вплинути на проєкт. (рис.6.9)

2. Чому SWOT важливий у проєктах?

✓ Дає цілісне бачення середовища проєкту.

✓ Допомагає визначати джерела ризиків (слабкі сторони й загрози).

✓ Дозволяє використовувати сильні сторони та можливості як захисні механізми проти ризиків.

✓ Є простим і зрозумілим інструментом для обговорення зі стейкхолдерами.

S (Сильні сторони)	W (Слабкі сторони)
Досвідчена команда розробників	Висока залежність від одного замовника
Використання сучасного стеку технологій	Недостатня кількість тестувальників
Налагоджені процеси CI/CD	Обмежений бюджет
O (Можливості)	**T (Загрози)**
Розширення на міжнародні ринки	Зміни в законодавстві (GDPR, податки)
Попит на мобільні застосунки	Поява сильного конкурента
Співпраця з великими корпораціями	Економічна нестабільність, війна

Рисунок 6.9 – Приклад SWOT-аналізу для ІТ-проєкту

4. Як SWOT допомагає у ризик-менеджменті?

✓ Weaknesses → План розвитку (покращення процесів, найм персоналу).

✓ Threats → Risk Register (внесення у реєстр і визначення стратегії реагування).

✓ Strengths + Opportunities → Захист (як компенсувати ризики).

Висновок. SWOT-аналіз – це базовий, але дуже корисний метод у ризик-менеджменті. Він дозволяє не тільки виявити загрози, а й побачити, як внутрішні сильні сторони та зовнішні можливості можуть мінімізувати ризики.

6.5.4. Метод Монте-Карло для оцінки невизначеності

1. Суть методу Монте-Карло

Метод Монте-Карло – це статистичний підхід, який використовує багаторазове моделювання сценаріїв проєкту з випадковими вхідними даними. Результат дозволяє оцінити ймовірність успішного завершення проєкту в певні строки та бюджет.

2. Як це працює?

✓ Визначаються ключові змінні (наприклад: тривалість завдань, вартість ресурсів, кількість помилок).

✓ Для кожної змінної задається діапазон можливих значень (мінімум, максимум, найбільш імовірне).

✓ Генерується тисячі симуляцій із випадковими варіаціями.

✓ Аналізуються результати → будується розподіл ймовірностей для строків, витрат чи інших параметрів.

3. Приклад у IT-проєкті (рис.6.10).

Проєкт має завдання з такими строками (у днях):

✓ Завдання A: 5-8 днів

✓ Завдання B: 3-6 днів

✓ Завдання C: 4-7 днів

Після симуляцій (10 000 прогонів):

✓ Ймовірність завершення у 15 днів ≈ 40 %

✓ Ймовірність завершення у 18 днів ≈ 75 %

✓ Ймовірність завершення у 20 днів ≈ 95 %

Таким чином, керівник може планувати більш реалістичні дедлайни.

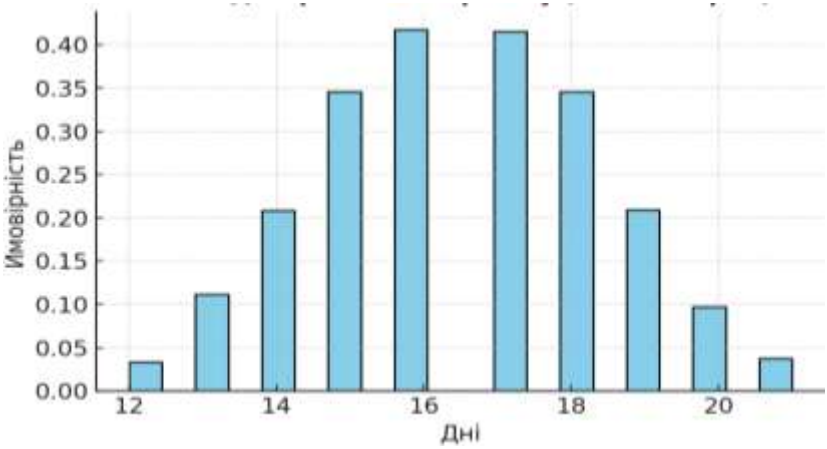

Рисунок 6.10 – Розподіл тривалості проєкту (Монте-Карло)

4. Переваги методу:

✓ Дає кількісну оцінку ризиків.

✓ Дозволяє бачити не лише «середній сценарій», а й крайні варіанти.

✓ Допомагає у фінансовому й часовому плануванні.

5. Недоліки:

✓ Вимагає якісних вхідних даних (оцінки експертів, історичні дані).

✓ Потребує спеціальних інструментів (MS Project, @Risk, Python).

✓ Може бути складним для пояснення нетехнічним стейкхолдерам.

Висновок. Метод Монте-Карло дозволяє перетворити невизначеність у цифри, що суттєво підвищує якість прийняття управлінських рішень. Він особливо корисний у великих ІТ-проєктах із високою складністю й залежностями.

6.5.5. Аналіз чутливості (Sensitivity Analysis)

1. Суть методу

Аналіз чутливості дозволяє визначити, які фактори найбільше впливають на результат проєкту. Ідея проста: змінюємо один параметр → дивимося, як це впливає на фінальний результат (час, бюджет, якість).

2. Використання в управлінні ризиками

✓ Дає можливість зрозуміти, які змінні створюють найбільші ризики.

✓ Дозволяє зосередити управлінські зусилля на найкритичніших параметрах.

✓ Використовується для прийняття рішень: на що витратити більше часу й ресурсів у моніторингу.

3. Приклад у IT-проєкті

Уявимо, що ми оцінюємо вартість розробки мобільного застосунку. На бюджет впливають:

✓ Тривалість розробки (тижні)

✓ Кількість розробників

✓ Ставка розробника за годину

Після аналізу чутливості виявляється:

✓ Збільшення тривалості на 10 % → +12 % до бюджету.

✓ Збільшення ставки на 10 % → +10 % до бюджету.

✓ Зменшення продуктивності команди на 10 % → +20 % до бюджету.

Висновок: продуктивність команди є найбільш критичним фактором ризику.

4. Інструменти для аналізу

✓ Excel/Google Sheets – найпростіший варіант (what-if analysis, сценарії).

✓ MS Project – для великих проєктів.

✓ Python/R – для більш гнучких симуляцій.

5. Візуалізація результатів

Часто використовують "торнадо-діаграми" (tornado charts), де показано вплив кожного фактору на результат у відсотках (рис.6.11).

Висновок. Аналіз чутливості дозволяє сфокусувати управління ризиками на найважливіших факторах. Це допомагає уникнути ситуації, коли команда витрачає ресурси на другорядні ризики, і підвищує ефективність моніторингу.

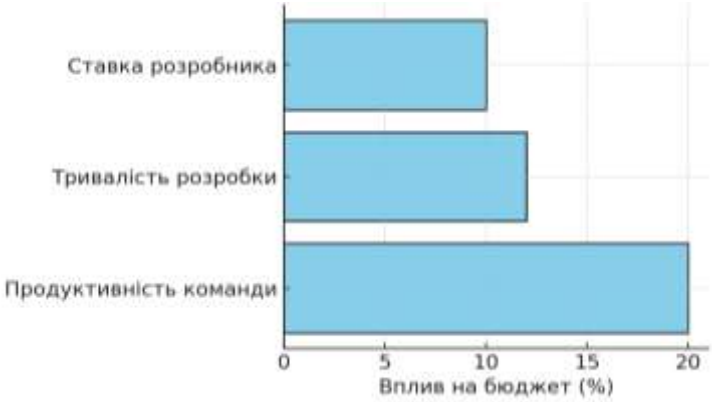

Рисунок 6.11 – Аналіз чутливості (торнадо-діаграма)

6.5.6. Методи реагування на ризики

1. Уникання ризику (Risk Avoidance)

Суть: повністю усунути ризик або змінити план так, щоб він не виник.

Приклад: відмова від використання нестабільної технології на користь перевіреної.

2. Зменшення ризику (Risk Mitigation / Reduction)

Суть: знизити ймовірність або вплив ризику.

Приклад: проведення додаткового тестування для зменшення кількості багів.

3. Передача ризику (Risk Transfer)

Суть: перекласти відповідальність на іншу сторону.

Приклад: укладення контракту з хостинг-провайдером із гарантіями SLA.

4. Прийняття ризику (Risk Acceptance)

Суть: свідомо прийняти ризик і бути готовим до наслідків.

Приклад: команда погоджується на можливі затримки через складність інтеграції, закладаючи резерв часу.

5. Вибір методу

Залежить від критичності ризику (визначається через матрицю ризиків).

Для низьких ризиків зазвичай застосовують прийняття.

Для середніх – зменшення.

Для високих – уникання або передача.

Таблиця 6.13 – Приклади для кожного методу

Метод	Суть	Приклад у IT-проєкті
Уникання (Avoidance)	Повністю усунути ризик або змінити план, щоб він не виник	Відмова від використання експериментальної бібліотеки, яка може спричинити баги
Зменшення (Mitigation)	Знизити ймовірність або вплив ризику	Впровадження автоматизованого тестування, щоб зменшити кількість помилок у коді
Передача (Transfer)	Перекласти відповідальність за ризик на іншу сторону	Заключення договору з хмарним провайдером із гарантією SLA для безперервної роботи сервісу
Прийняття (Acceptance)	Свідомо прийняти ризик і бути готовим до його наслідків	Команда погоджується на можливі затримки через складну інтеграцію й закладає резерв часу

6.6. Загальні висновки 6

6.6.1. Загальні висновки

1. Ресурси в IT-проєкті поділяються на людські, фінансові, матеріальні та технічні.

2. Планування ресурсів дозволяє уникнути перевантаження команди та нестачі бюджету.

3. Інструменти управління часом (діаграма Ганта, метод критичного шляху) допомагають виявити залежності між завданнями й визначити вузькі місця.

4. Типи ризиків включають технічні, бізнесові, організаційні та зовнішні, кожен з яких має свої приклади в IT.

5. Для аналізу ризиків застосовуються різні методи:

✓ Risk Register

- ✓ Risk Matrix
- ✓ SWOT-аналіз
- ✓ Метод Монте-Карло
- ✓ Аналіз чутливості

6. Методи реагування на ризики: уникання, зменшення, передача, прийняття.

7. Ефективне управління ресурсами та ризиками підвищує стійкість проєкту та ймовірність його успішного завершення.

6.6.2. Питання для самоперевірки

1. Які основні види ресурсів використовуються в ІТ-проєктах?
2. У чому різниця між діаграмою Ґанта та методом критичного шляху?
3. Наведить приклади технічних і бізнесових ризиків у проєкті.
4. Для чого використовується Risk Register?
5. Як інтерпретується Risk Matrix? Які категорії ризиків вона відображає?
6. Які елементи включає SWOT-аналіз?
7. Як працює метод Монте-Карло для оцінки строків і бюджету проєкту?
8. У чому переваги та недоліки аналізу чутливості?
9. Які чотири основні методи реагування на ризики?
10. Який метод реагування доцільно застосовувати для низьких ризиків?
11. Який метод найчастіше обирають для критичних ризиків?
12. Чому управління ризиками важливе для стейкхолдерів?

6.6.3. Завдання для самостійної роботи

1. Побудувати спрощену діаграму Ґанта для навчального проєкту (наприклад, створення мобільного застосунку).
2. Скласти приклад Risk Register для 5 можливих ризиків у цьому проєкті.
3. Виконати SWOT-аналіз своєї навчальної команди або проєкту.
4. Визначити стратегії реагування на кожен з ризиків.

6.7. Контрольні питання

1. Що таке управління ресурсами в IT-проєкті? Які категорії ресурсів виділяють?
2. Які наслідки має неправильний розподіл ресурсів у IT-проєкті? Наведіть реальний або гіпотетичний приклад.
3. Що таке «overbooking» ресурсів і чому він є однією з типових помилок у плануванні IT-проєктів?
4. Як пов'язані між собою управління ресурсами та управління ризиками? Проілюструйте взаємозв'язок.
5. Що таке ризик у контексті IT-проєкту? Чим він відрізняється від проблеми (issue)?
6. Назвіть і охарактеризуйте основні процеси управління ризиками: ідентифікація, оцінка, реагування, моніторинг.
7. Що таке матриця ризиків (probability/impact matrix)? Як вона використовується для пріоритизації ризиків?
8. Які стратегії реагування на ризики існують (уникнення, пом'якшення, передача, прийняття)? Наведіть приклади для кожної.
9. Що таке резервний план (contingency plan) і коли він застосовується в IT-проєктах?
10. Які технічні ризики найчастіше зустрічаються в IT-проєктах і як їх можна мінімізувати?
11. Як вигорання (burnout) команди є ресурсним ризиком? Які заходи запобігають цьому явищу?
12. Наведіть кейс проєктного провалу через погане управління ресурсами або ризиками. Проаналізуйте причини.
13. Що таке бюджетний резерв (contingency reserve і management reserve) у проєктному менеджменті?
14. Як управляти ризиками в Agile-проєктах на відміну від традиційного підходу?
15. Що таке реєстр ризиків (risk register)? Опишіть його структуру та порядок ведення.

7. УПРАВЛІННЯ ЗМІНАМИ В ІТ-ПРОЄКТАХ

7.1. Вступ

Зміни в ІТ-проєктах є не винятком, а нормою. Будь-який проєкт починається з певного бачення та набору вимог, однак у процесі роботи ситуація рідко залишається стабільною.

7.1.1. Чому зміни є невід'ємною частиною ІТ-проєктів?

ІТ-індустрія відзначається високою динамікою: технології швидко розвиваються, з'являються нові інструменти, а бізнес-потреби змінюються під тиском ринку та конкурентів.

Наприклад:

✓ Нові технології (нові версії мов програмування, фреймворків або хмарних сервісів) можуть зробити початково обрані рішення застарілими.

✓ Зміни у бізнес-цілях – компанія може переорієнтувати продукт на інший сегмент клієнтів.

✓ Регуляторні вимоги – впровадження нових стандартів (наприклад, GDPR або локальних законів про захист даних) часто змушує переглядати архітектуру системи.

✓ Зворотний зв'язок від користувачів під час тестування чи бета-релізу також ініціює нові зміни, які не були враховані на старті.

Таким чином, управління змінами – це не факультативна опція, а ключова компетенція будь-якого менеджера проєкту та бізнес-аналітика (рис.7.1). Уміння адаптуватися до нових умов відрізняє успішний проєкт від провального.

Рисунок 7.1 – Зв'язок між вимогами, змінами та ризиками

Ця схема демонструє, що зміни часто виникають із нових або уточнених вимог, а кожна зміна може породжувати нові ризики для проєкту.

7.1.2. Вартість і наслідки неконтрольованих змін

1. Чому зміни неминучі?

У будь-якому ІТ-проєкті виникають зміни: нові бізнес-вимоги, технічні оновлення, зміна ринкової ситуації. Але коли ці зміни відбуваються без контролю, проєкт стикається з серйозними ризиками.

2. Вартість неконтрольованих змін (табл.7.1):

✓ Фінансова вартість: збільшення бюджету через переробки, додаткових спеціалістів або ліцензій.

✓ Часова вартість: відставання від графіку, зрив релізів.

✓ Людський фактор: вигорання команди, падіння мотивації через «хаотичні правки».

✓ Технічна вартість: поява технічного боргу, складність інтеграцій, нестабільний продукт.

Таблиця 7.1 – Наслідки неконтрольованих змін

Тип вартості	Приклад	Наслідки
Фінансова	Додаткові витрати на переробку та ліцензії	Зростання бюджету
Часова	Відставання від графіку, зрив релізів	Затримки проєкту
Людська	Перевантаження, постійні зміни пріоритетів	Вигорання, падіння мотивації
Технічна	Непередбачувані інтеграції, переробки коду	Технічний борг, нестабільність системи

3. Наслідки для бізнесу

✓ Продукт виходить на ринок пізніше, ніж конкуренти.

✓ Якість рішення знижується, бо команда працює у «пожежному режимі».

✓ Замовники втрачають довіру до команди та постачальника.

✓ Зростає ймовірність провалу всього проєкту.

4. Приклад

Неконтрольовані зміни →

↑ Бюджет ↑ Час ↓ Якість ↓ Мотивація

= Ризик провалу проєкту

7.1.3. Зв’язок між змінами, ризиками та вимогами

1. Взаємозалежність

✓ Вимоги – це те, що замовник і користувачі очікують від продукту.

✓ Зміни – неминучі, бо вимоги рідко залишаються статичними (ринок, конкуренція, нові технології).

✓ Ризики – з’являються щоразу, коли зміни впроваджуються без достатнього аналізу або контролю.

Таким чином, зміни → впливають на вимоги → створюють нові ризики, якщо їх не управляти системно (рис.7.2).

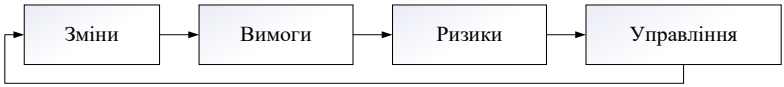

Рисунок 7.2 – Логіка зв’язку

2. Приклади взаємозв’язку:

✓ Зміна у вимогах: клієнт просить інтегрувати нову платіжну систему.

✓ Ризик: несумісність із поточною архітектурою → можливі збої та додаткові витрати.

✓ Наслідок: затримка релізу, збільшення бюджету.

3. Практичний висновок

Управління вимогами та ризиками невіддільне від управління змінами. Успіх проєкту залежить від того, наскільки чітко команда вміє:

✓ відстежувати зміни,

✓ оцінювати їхній вплив на вимоги,

✓ прогнозувати ризики.

7.2. Причини змін у проєктах

Бізнес-середовище сьогодні є надзвичайно динамічним. Те, що було актуальним на момент старту проєкту, може змінитися вже через кілька місяців. Тому зміни у бізнес-вимогах – один із найпоширеніших драйверів трансформацій в ІТ-проєктах.

7.2.1. Зміни у бізнес-вимогах і пріоритетах

Основні причини зміни бізнес-вимог:

1. Ринкові умови. Зміни у поведінці споживачів, поява нових конкурентів або нових бізнес-моделей змушують компанію швидко реагувати.

2. Стратегічні цілі компанії. Якщо організація вирішує вийти на новий сегмент ринку, продукт доведеться адаптувати.

3. Зворотний зв'язок від користувачів. Після тестування MVP або першої версії продукту часто виникають нові пріоритети – наприклад, користувачі вимагають мобільну версію раніше за веб.

4. Фінансові чинники. Обмеження бюджету або нові інвестиції можуть змінити масштаби та фокус проєкту.

Приклад із практики.

Фінтех-стартап запускає платформу для онлайн-платежів. Спочатку основний акцент робився на B2B-клієнтів. Але після тестування ринку компанія отримала значний інтерес від кінцевих користувачів (B2C). Це призвело до зміни бізнес-пріоритетів: з’явилася потреба у швидкій розробці мобільного застосунку для масового сегменту (табл.7.2).

Таблиця 7.2 – Приклади

Ситуація	Зміна у вимогах	Наслідок для проєкту
Новий конкурент вивів продукт із унікальним функціоналом	Необхідність додати аналогічну функцію	Збільшення строків і бюджету
Після MVP користувачі вимагають мобільну версію	Зміщення фокусу з вебплатформи на мобільний застосунок	Перерозподіл ресурсів, можливі затримки
Компанія отримала нове фінансування	Розширення функціоналу продукту	Необхідність перегляду дорожньої карти

Висновок. Зміни у бізнес-вимогах – це природна реакція на нові можливості й виклики ринку. Завдання проєктного менеджера та бізнес-аналітика полягає у тому, щоб правильно зафіксувати ці зміни, оцінити їхній вплив на час, бюджет і обсяг робіт, та уникнути хаотичного

«перекроювання» проєкту.

7.2.2. Технологічні інновації як причина змін

Технології в ІТ розвиваються швидше, ніж у більшості інших галузей. Те, що сьогодні є стандартом, уже завтра може застаріти. У результаті команди стикаються з необхідністю адаптувати свої проєкти до нових технічних можливостей.

Основні приклади технологічних інновацій, що викликають зміни:

1. Оновлення фреймворків та мов програмування. Наприклад, вихід нової версії Angular чи React, яка кардинально змінює підхід до розробки.
2. Хмарні технології. Перехід компаній на AWS, Azure чи GCP може вимагати перенесення архітектури продукту.
3. Штучний інтелект і автоматизація. Багато компаній сьогодні додають AI-модулі у свої продукти, навіть якщо спочатку цього не планувалося.
4. Кібербезпека. Зміна стандартів безпеки (наприклад, необхідність підтримки нових алгоритмів шифрування) змушує переглядати код і процеси.
5. Інтеграції. Поява нових API або популярних платформ (наприклад, інтеграція з WhatsApp Business API) змінює функціональні вимоги.

Приклад із практики:

Компанія розробляла корпоративну систему управління персоналом на застарілому стеку технологій. У середині проєкту було вирішено перейти на мікросервісну архітектуру з використанням Kubernetes. Це спричинило значну перебудову плану, але дало системі масштабованість і гнучкість у майбутньому (табл.7.3).

Висновок. Технологічні інновації – один із ключових факторів змін у проєктах. Вони можуть підвищити конкурентоспроможність продукту, але водночас збільшують витрати та потребують ретельного управління змінами.

Таблиця 7.3 – Приклади технологічних інновацій та їх вплив

Технологічна інновація	Можлива зміна у проєкті	Наслідок
Вихід нової версії фреймворку (React, Angular)	Необхідність оновлення архітектури та коду	Затримки у розробці, підвищення продуктивності у майбутньому
Перехід на хмарні рішення (AWS, Azure, GCP)	Міграція даних та перебудова архітектури	Краще масштабування, додаткові витрати на адаптацію
Додавання AI/ML-модулів	Розробка нових алгоритмів та інтеграція моделей	Зростання вартості, підвищення конкуренто-спроможності продукту
Нові стандарти кібербезпеки	Оновлення протоколів шифрування та процесів доступу	Захист даних, додаткові витрати на перевірку системи
Інтеграція з новими API (наприклад, WhatsApp Business)	Додавання інтеграційного модуля	Розширення функціоналу, збільшення строків

7.2.3. Регуляторні та юридичні вимоги

IT-продукти часто працюють у середовищі, яке регулюється законами та міжнародними стандартами. Будь-яка зміна нормативних актів або введення нових стандартів безпеки може вимагати адаптації продукту.

Основні приклади (табл.7.4):

1. Захист персональних даних. Введення GDPR у ЄС змусило тисячі компаній змінити архітектуру своїх систем та процеси зберігання даних.

2. Локальні закони. Наприклад, вимога зберігати дані користувачів тільки на серверах у межах певної країни.

3. Фінансове регулювання. Банківські та фінтех-проєкти повинні дотримуватися вимог PCI DSS, що впливає на обробку та зберігання даних платіжних карток.

4. Стандарти кібербезпеки. ISO/IEC 27001 або NIST можуть вимагати додаткових процесів моніторингу та контролю доступу.

5. Вимоги до доступності. Уряди деяких країн зобов’язують цифрові продукти дотримуватися стандартів WCAG для людей із вадами зору/слуху.

Таблиця 7.4 – Приклади вимог і вплив на ІТ-продукти

Регуляторна вимога / стандарт	Сфера	Що змінюється у продукті	Приклади змін у процесах	Ризик невиконання
GDPR / ePrivacy	Персональні дані (ЄС)	Механізм згоди (consent), право на видалення, портативність даних.	DPIA (оцінка впливу на приватність), реєстр обробки даних, процеси обробки запитів суб'єктів.	Штрафи, блокування сервісу в ЄС, репутаційні втрати.
Закон про захист персональних даних	Персональні дані (нац. юрисдикція)	Локалізація даних, згоди, політика збереження даних.	Призначення відповідального (DPO), реєстрація баз, внутрішні регламенти.	Штрафи регулятора, обмеження діяльності.
PCI DSS	Платіжні картки / фінтех	Сегментація мережі, шифрування PAN, токенізація.	Регулярні сканування вразливостей, pen-тести, логування доступів.	Втрата права приймати платежі, штрафи банків-еквайрів.
ISO/IEC 27001 / NIST	Інформаційна безпека	Контроль доступу, управління ключами, журналювання.	ISMS (система керування безпекою), аудит, політики безпеки.	Інциденти безпеки, витоки даних, штрафи замовників.
WCAG 2.1+	Доступність (Accessibility)	ARIA-атрибути, контрастність, навігація з клавіатури.	UX-аудит доступності, тестування з користувачами з ООП.	Скарги користувачів, державні санкції у публічному секторі.
Data Localization	Локалізація даних	Зберігання даних у конкретній країні/регіоні	Вибір дата-центрів, оновлення політик обробки даних.	Правові санкції, неможливість оперувати на ринку.

Приклад із практики .

Міжнародна компанія розробляла SaaS-рішення для обробки клієнтських даних. Після введення GDPR довелося додати функції видалення даних на запит користувача та перебудувати систему згоди на обробку даних. Це змінило як бекенд-архітектуру, так і UX-прототипи. Нижче – структурований огляд типових вимог та їхнього впливу, матриця RACI змін наведена у табл.7.5.

Чекліст впливу на продукт і процеси:

✓ Дані: які категорії ПД ми збираємо/зберігаємо/передаємо?

✓ Юрисдикції: у яких країнах працює продукт і чи є вимоги локалізації?

✓ Згода та права суб'єктів: механізми отримання/відкликання, export/delete?

✓ Безпека: шифрування, контроль доступу, журналювання, моніторинг інцидентів.

✓ Процеси: хто власник регламентів, як проводимо DPIA/аудит, як обробляємо запити?

✓ Постачальники: чи відповідають провайдери (CSP, процесинг) вимогам? SLA/додаткові угоди.

Таблиця 7.5 – RACI для змін, спричинених регуляціями

Активність	PM	BA	Legal/DPO	Security/DevOps
Моніторинг регуляцій	C	C	R/A	I
Оцінка впливу (DPIA/Risk)	A	R	C	C
Підготовка Change Request	A	R	C	C
Реалізація змін	A	C	I	R
Аудит/звітність	I	C	R/A	C

Схема процесу комплаєнсу наведена на рис.7.3.

Рисунок 7.3 – Процес комплаєнсу

Висновок. Регуляторні та юридичні зміни часто не залежать від команди чи замовника, але їх ігнорування призведе до юридичних санкцій, штрафів та втрати клієнтів. Тому управління змінами має включати моніторинг законодавчих і стандартних вимог.

7.2.4. Зовнішні фактори: конкуренція, ринок, форс-мажори

Не всі зміни в ІТ-проєктах виникають зсередини компанії. Існує низка зовнішніх факторів, які можуть кардинально вплинути на продукт і стратегію його розвитку:

1. Конкуренція.

✓ Поява нового гравця з унікальним функціоналом змушує швидко адаптувати дорожню карту продукту.

✓ Конкуренти можуть знизити ціни або запропонувати нові бізнес-моделі (наприклад, freemium), що призводить до зміни пріоритетів у власному проєкті.

2. Ринок.

✓ Технологічні тренди (наприклад, вибухове зростання популярності мобільних додатків чи AI-рішень) впливають на те, куди інвестуються ресурси.

✓ Попит з боку користувачів може змінюватися залежно від економічної ситуації: у кризові періоди більше цінуються продукти, що оптимізують витрати.

3. Форс-мажори.

✓ Пандемії, війни, кібератаки, перебої в енергопостачанні можуть призвести до повної перебудови процесів.

✓ У таких випадках змінюються навіть базові умови роботи: дистанційний формат, зміни у графіках постачання, перенесення інфраструктури в інші регіони.

Приклад із практики.

Під час пандемії COVID-19 багато компаній були змушені за лічені тижні впровадити дистанційну роботу та додати функціонал онлайн-співпраці в продукти, які спочатку не були для цього призначені.

Таблиця 7.6 – Приклади зовнішніх факторів та їхній вплив

Зовнішній фактор	Приклад ситуації	Вплив на проєкт
Конкуренція	Новий гравець випустив дешевший продукт з аналогічним функціоналом	Необхідність змінити цінову політику або додати нові фічі
Ринок	Популярність AI-рішень різко зросла	Перенесення акценту на інтеграцію AI у власний продукт
Ринок	Зниження купівельної спроможності користувачів через економічну кризу	Фокус на функціях, що оптимізують витрати клієнтів
Форс-мажор	Пандемія COVID-19. Війна в Україні	Необхідність додати функції для дистанційної співпраці, перевести команду на remote
Форс-мажор	Війна, перебої в енергопостачанні	Перенесення серверів у інші регіони, перебудова процесів підтримки

Висновок. Зовнішні фактори неможливо контролювати, але їх потрібно враховувати у плануванні. Саме для цього в управлінні проєктами застосовують аналіз ризиків і сценарне планування, щоб мати запасні варіанти дій у випадку непередбачуваних подій.

7.2.5. Приклади з практики: зміна API, потреба інтеграції з новими сервісами

ІТ-проєкти часто залежать від зовнішніх систем, сервісів чи бібліотек. Будь-які зміни з боку цих залежностей можуть спричинити серйозні зміни у самому продукті.

Типові ситуації:

1. Зміна API стороннього сервісу.

Компанія інтегрує свій продукт із платіжним шлюзом. Постачальник

оновлює API, змінює методи автентифікації чи формат даних.

Результат: потрібно переробити частину коду, внести зміни у тестування та документацію.

2. Відмова від підтримки старої версії API.

Наприклад, Google Maps або Facebook API припиняють підтримку старих версій.

Результат: команда змушена терміново оновлювати інтеграцію, щоб продукт не втратив критичні функції.

3. Потреба інтеграції з новими сервісами.

Замовник вирішує, що продукт має підтримувати інтеграцію з популярними месенджерами чи CRM-системами.

Результат: з'являється новий обсяг робіт, збільшується складність тестування.

4. Зміна умов використання сторонніх сервісів (табл.7.7).

Наприклад, платний тариф API, який раніше був безкоштовним.

Результат: зміна бюджету та, можливо, пошук альтернативних рішень.

Таблиця 7.7 – Приклади змін через зовнішні сервіси

Ситуація	Вплив на продукт	Наслідок для проєкту
Оновлення API платіжного шлюзу	Зміна методів автентифікації або формату даних	Необхідність переробки коду, тестів та документації
Відмова від підтримки старої версії API (Google Maps, Facebook)	Стара інтеграція перестає працювати	Термінове оновлення інтеграції, ризик зупинки функцій
Інтеграція з новими сервісами (CRM, месенджери)	Додавання нового функціоналу для користувачів	Збільшення обсягу робіт і складності тестування
Зміна тарифів або політики доступу до API	Раніше безкоштовний сервіс стає платним	Збільшення бюджету, потреба у пошуку альтернатив

Приклад. Компанія-стартап інтегрувала свій продукт із Twitter API для збору аналітики. Після зміни політики доступу Twitter у 2023 році команда була змушена змінити підхід: перейти на платний тариф і інтегрувати альтернативні джерела даних, що призвело до зміни архітектури та бюджету.

Висновок. Залежність від зовнішніх сервісів – це завжди ризик для проєкту. Тому управління змінами передбачає:

- ✓ моніторинг зовнішніх залежностей;
- ✓ план «В» на випадок відмови чи зміни стороннього сервісу;
- ✓ гнучку архітектуру, яка дозволяє швидко інтегрувати альтернативи.

7.3. Процеси управління змінами

Управління змінами проєкту залежить від моделі управління та складається з декілька фаз.

7.3.1. Change Management як частина PMBOK та Agile-підходів

1. У PMBOK (Project Management Body of Knowledge).

Управління змінами розглядається як структурований процес, який проходить через Change Control System. Основна мета – контролювати будь-які зміни до обсягу робіт (scope), бюджету, графіку та якості.

PMBOK визначає Change Control Board (CCB) – орган (комітет), який ухвалює рішення про прийняття або відхилення змін. Усі зміни повинні бути задокументовані у вигляді Change Request. Це забезпечує формалізований підхід, зручний для великих проєктів із високим рівнем відповідальності (державні програми, великі корпорації).

2. В Agile-підходах (Scrum, Kanban).

Change Management має іншу природу: зміни сприймаються як природна частина процесу. Замість формальних заявок зміни вносяться через беклог продукту (Product Backlog).

Пріоритизацією займається Product Owner, який вирішує, які зміни потраплять у наступний спринт. Agile орієнтується на гнучкість та швидке реагування, тому немає окремої ради змін – команда сама інтегрує нові вимоги у процес. Це підходить для динамічних середовищ, де важлива швидкість (стартапи, продуктові компанії).

Таблиця 7.8 – Порівняння PMBOK і Agile у Change Management.

Критерій	PMBOK	Agile
Формалізація	Висока, через Change Control Board	Низька, зміни одразу йдуть у беклог
Швидкість реакції	Низька-середня (через погодження)	Висока (пряме оновлення беклогу)
Документація	Обов'язкові Change Requests	Мінімальна документація
Ролі	PM, CCB, BA	Product Owner, команда
Сфера застосування	Великі, регульовані проєкти	Динамічні, інноваційні середовища

Висновок. PMBOK і Agile пропонують два різні підходи до управління змінами:

- ✓ PMBOK – контроль і передбачуваність.
- ✓ Agile – швидкість і гнучкість.

У сучасних проєктах часто використовують гібридні моделі, де стратегічні зміни йдуть за PMBOK, а тактичні – через Agile-процеси.

7.3.2. Основні етапи управління змінами

Етапи процесу управління змінами

1. Ідентифікація потреби у зміні.
2. Аналіз впливу зміни (cost, time, scope, quality).
3. Прийняття рішення (Steering Committee, Change Control Board).
4. Планування та реалізація змін.
5. Відстеження та документування результатів.

7.3.2.1. Основні етапи: 1. Ідентифікація потреби у зміні

Перший крок у процесі управління змінами – це виявлення того, що в проєкті щось необхідно змінити. Цей етап критично важливий, адже саме тут формується усвідомлення проблеми або нової можливості.

Джерела потреби у змінах (табл.7.9):

- ✓ Бізнес-вимоги. Наприклад, зміна пріоритетів замовника чи ринку.

✓ Технологічні чинники. Вихід нових версій фреймворків, поява нових стандартів безпеки.

✓ Зворотний зв'язок від користувачів. Нові очікування або виявлені недоліки під час тестування.

✓ Регуляторні вимоги. Нові закони чи стандарти (GDPR, PCI DSS).

✓ Внутрішні проблеми проєкту. Перевищення бюджету, затримки, технічні обмеження.

Таблиця 7.9 – Джерела змін і ініціатори

Джерело зміни	Приклад	Хто ініціює
Бізнес-вимоги	Замовник змінив цільову аудиторію продукту	Stakeholder / BA
Технологічні чинники	Вихід нової версії фреймворку	Dev Team / Architect
Зворотний зв'язок	Користувачі вимагають авторизацію через Google	Кінцеві користувачі / BA
Регуляторні вимоги	Впровадження GDPR	Legal / Compliance / BA
Внутрішні проблеми	Перевищення бюджету	PM / Finance

Хто може ініціювати зміни:

✓ Замовник (Stakeholder).

✓ Бізнес-аналітик (BA).

✓ Project Manager (PM).

✓ Технічна команда (розробники, QA, DevOps).

✓ Кінцеві користувачі (через відгуки).

Приклад:

Під час тестування мобільного додатку команда отримала зворотний зв'язок: користувачі очікують функцію авторизації через Google і Facebook. Хоча спочатку ця функція не входила у вимоги, вона стає критичною для підвищення зручності користування. Таким чином формується потреба у зміні.

Висновок. На цьому етапі не робиться остаточних рішень, а лише фіксується факт потреби. Результатом повинно бути оформлення ініціативи

зміни (Change Request) або занесення пропозиції до беклогу продукту (в Agile).

7.3.2.2. Основні етапи: 2. Аналіз впливу зміни

Після того як потребу у зміні зафіксовано, необхідно оцінити її вплив на проєкт. Це один із найважливіших етапів, адже без якісного аналізу зміна може призвести до непередбачуваних наслідків.

Що оцінюється при аналізі (табл.7.10):

1. Scope (обсяг робіт):

✓ Які частини продукту зачепить зміна?

✓ Чи потрібно додатково змінювати вимоги, архітектуру, тестові сценарії?

2. Cost (вартість).

✓ Чи потребує зміна додаткових фінансових ресурсів?

✓ Чи збільшить вона витрати на інфраструктуру чи ліцензії?

3. Time (час).

✓ Наскільки зміна вплине на графік?

✓ Чи потрібно переносити дедлайни?

4. Quality (якість).

✓ Чи може зміна вплинути на стабільність продукту?

✓ Які додаткові ризики для тестування виникають?

5. Risks (ризики).

✓ Чи створює зміна нові загрози?

✓ Чи підвищує вона складність підтримки?

Методи аналізу:

✓ Impact Analysis – системний аналіз впливу змін на всі артефакти проєкту (вимоги → код → тест-кейси → документація).

✓ Cost-Benefit Analysis – співставлення витрат із потенційною вигодою.

✓ What-if Scenarios – сценарне моделювання: що станеться, якщо зміна буде впроваджена або відхилена.

Таблиця 7.10 – Фактори для аналізу

Фактор	Що аналізується	Типові запитання
Scope (обсяг робіт)	Які вимоги, модулі, сценарії будуть змінені?	Чи потрібно змінювати архітектуру, тест-кейси, документацію?
Cost (вартість)	Фінансовий вплив зміни	Чи потребує додаткового бюджету? Чи є нові витрати на інфраструктуру або ліцензії?
Time (час)	Вплив на графік та дедлайни	Наскільки зміна зсуває терміни? Чи потрібен додатковий спринт?
Quality (якість)	Стабільність і продуктивність продукту	Чи вплине зміна на тестування або надійність системи?
Risks (ризики)	Нові можливі загрози для проєкту	Чи створює зміна нові ризики? Чи ускладнює підтримку?

Приклад із практики.

Команда отримала запит на інтеграцію з новою CRM-системою. Аналіз показав, що:

- ✓ потрібно змінити бекенд-архітектуру;
- ✓ строки збільшаться на 2 спринти;
- ✓ бюджет зросте на 15 %;
- ✓ але вигода – розширення клієнтської бази і збереження ключового замовника.

Висновок. Аналіз впливу допомагає приймати зважені рішення. Він забезпечує прозорість і дає змогу замовнику зрозуміти наслідки кожної зміни до того, як вона буде реалізована.

7.3.2.3. Основні етапи: 3. Прийняття рішення щодо зміни

Після того як вплив зміни був проаналізований, настає момент ухвалення рішення: приймати зміну чи відхиляти її.

1. У класичному підході (PMBOK):

Існує спеціальний орган – Change Control Board (CCB), який розглядає всі Change Requests. Рішення базується на аналітиці: вплив на обсяг, строки,

бюджет, ризики.

Можливі варіанти:

- ✓ Прийняти зміну (у повному обсязі).
- ✓ Відхилити зміну (з аргументацією).
- ✓ Відкласти зміну (наприклад, перенести у майбутні фази).
- ✓ Модифікувати зміну (адаптувати, зменшити її масштаб).

2. В Agile-підходах:

Рішення приймається у процесі беклог-менеджменту (табл.7.11). Власник продукту (Product Owner) спільно з командою визначає, чи варто включати зміну в найближчий спринт. Зміна може бути одразу відображена у Product Backlog або навіть потрапити до Sprint Backlog. Рішення ухвалюється швидше, без довгих погоджень.

Критерії для ухвалення рішення:

- ✓ Відповідність стратегічним цілям продукту.
- ✓ Реалістичність з точки зору ресурсів.
- ✓ Баланс «вартість/вигода».
- ✓ Ризики від впровадження чи відмови від зміни.

Таблиця 7.11 – Варіанти рішень

Варіант рішення	Коли застосовується	Приклад
Прийняти зміну	Зміна критично важлива для досягнення цілей продукту	Додати функцію авторизації через Google для підвищення зручності
Відхилити зміну	Зміна суперечить стратегії або занадто дорога	Додаткові фічі, які не приносять бізнес-цінності
Відкласти зміну	Є корисною, але може зірвати дедлайни	Інтеграція push-сповіщень після основного релізу
Модифікувати зміну	Зміна занадто масштабна, її можна адаптувати	Замість повної CRM-інтеграції реалізувати базовий імпорт даних

Приклад із практики.

Команда аутсорсингової компанії отримала Change Request від клієнта

на впровадження push-сповіщень. Аналіз показав, що це потребує 2 додаткових тижні роботи і 10 % збільшення бюджету.

Вигода: підвищення залученості користувачів на 20 %.

Рішення ССВ: прийняти зміну, але інтегрувати у наступній фазі, щоб не зірвати поточний реліз.

Висновок. Прийняття рішення – це баланс між бажаннями клієнта, можливостями команди та стратегічними цілями продукту. Чіткий процес дозволяє уникнути хаотичних змін і зберегти контроль над проєктом.

7.3.2.4. Основні етапи: 4. Планування та реалізація змін

Після ухвалення рішення про прийняття зміни починається етап її планування та впровадження. Це практична частина, яка потребує чіткої координації.

Ключові кроки планування:

1. Визначення відповідальних.

✓ Хто виконує зміну? (розробники, QA, DevOps, BA).

✓ Хто контролює процес? (PM, Product Owner).

2. Розробка плану впровадження.

✓ Часові рамки (в який спринт або фазу включити зміну).

✓ Ресурси (додаткові люди, бюджет, обладнання).

✓ Пріоритетність (зміна може бути критичною або відкладеною).

3. Оновлення документації.

✓ У PMBOK: оновлення Project Management Plan та допоміжних планів.

✓ В Agile: оновлення Backlog і завдань у трекері (Jira, Trello, Azure DevOps).

4. Реалізація.

✓ Виконання робіт згідно з планом.

✓ Тестування змін (unit, integration, UAT).

✓ Поступове впровадження (rollout), щоб знизити ризики.

Практичні інструменти:

✓ Jira/Confluence – для опису та відстеження змін.

✓ CI/CD – для автоматизованого тестування та релізу.

✓ Feature Toggles – увімкнення/вимкнення нових функцій без ризику для стабільності.

Приклад із практики:

Компанія впроваджує зміну: нова система автентифікації з двофакторною перевіркою.

Планування: виділено окремий спринт, залучено 2 backend-розробників і 1 QA.

Реалізація: створено тестове середовище, проведено security-тести, поступово розгорнуто для 10 % користувачів.

Результат: зміна впроваджена без перебоїв у роботі сервісу.

Висновок. Ефективне планування та реалізація змін дозволяє мінімізувати ризики і гарантує, що зміни принесуть очікувану цінність без шкоди для основного продукту (табл.7.12).

Таблиця 7.12 – Схема процесу впровадження змін

Етап	Опис
Ініціація	Фіксація зміни (Change Request / Backlog)
Планування	Визначення строків, ресурсів, відповідальних
Виконання	Розробка та оновлення документації
Тестування	Unit, Integration, UAT
Впровадження	Rollout, контроль, моніторинг

7.3.2.5. Основні етапи: 5. Моніторинг і контроль змін

Навіть після того, як зміна була впроваджена, робота над нею не завершується. Потрібно постійно відстежувати її ефективність і контролювати вплив на проєкт (табл.7.13).

Завдання моніторингу:

1. Оцінка відповідності очікуванням.

✓ Чи принесла зміна ті результати, які планували?

✓ Чи зросла бізнес-цінність продукту?

2. Виявлення побічних ефектів.

✓ Чи виникли нові баги або проблеми з продуктивністю?

✓ Чи не постраждала безпека чи сумісність?

3. Вимірювання метрик.

✓ KPI: швидкість завантаження, кількість користувачів, рівень задоволеності клієнтів.

✓ ROI (Return on Investment) для бізнес-змін.

4. Зворотний зв'язок від користувачів.

✓ Agile-команди отримують відгуки під час Review або через системи аналітики.

✓ У Waterfall-підході – це може бути після User Acceptance Testing.

Інструменти моніторингу:

✓ Jira, Trello – відстеження виконання завдань.

✓ Google Analytics, Mixpanel – моніторинг поведінки користувачів.

✓ Grafana, Kibana – моніторинг продуктивності системи.

✓ Ретроспективи у Scrum – для оцінки ефективності змін.

Таблиця 7.13 – Об'єкт контролю, метрики та інструменти

Що контролюємо	Метрики	Інструменти
Відповідність очікуванням	Виконання вимог, рівень задоволеності	User Surveys, Scrum Review
Продуктивність системи	Час відгуку, аптайм, навантаження	Grafana, Kibana
Бізнес-ефективність	ROI, кількість нових клієнтів	Google Analytics, Power BI
Якість коду	Кількість багів, code coverage	SonarQube, Jira
Зворотний зв'язок	Оцінки, NPS, відгуки	Mixpanel, опитування користувачів

Приклад із практики.

Команда додала push-сповіщення у мобільний застосунок. Моніторинг показав, що рівень залученості користувачів зріс на 18 %. Водночас більшилася кількість скарг на «надмірну кількість повідомлень».

Рішення: налаштувати систему персоналізації, щоб користувачі самі обирали частоту сповіщень.

Висновок. Моніторинг і контроль змін дозволяє не лише впроваджувати нові функції, а й коригувати їх, щоб максимізувати вигоду та мінімізувати ризики. Це робить процес змін циклічним і безперервним.

7.3.2.6. Підсумок підрозділу

Процеси управління змінами – це структурований набір кроків, який дозволяє контролювати та впроваджувати зміни в ІТ-проєктах. Незалежно від підходу (PMBOK чи Agile) зміни проходять схожі етапи.

Основні етапи (рис.7.14):

1. Ідентифікація потреби – фіксація факту необхідності зміни.
2. Аналіз впливу – оцінка ефекту на обсяг, бюджет, строки, ризики.
3. Прийняття рішення – ухвалення рішення (прийняти, відхилити, відкласти, модифікувати).
4. Планування та реалізація – розробка плану впровадження та його виконання.
5. Моніторинг і контроль – перевірка результатів, відстеження побічних ефектів і корекція.

Таблиця 7.14 – Етапи управління змінами

Етап	Завдання	Результат
1. Ідентифікація потреби	Виявити причину зміни, зафіксувати Change Request	Оформлена пропозиція про зміну
2. Аналіз впливу	Оцінити вплив на scope, cost, time, quality, risks	Звіт з аналізу впливу
3. Прийняття рішення	CCB або Product Owner ухвалюють рішення	Затверджено або відхилено зміну
4. Планування та реалізація	Розробити план, виконати завдання, оновити документацію	Реалізована зміна
5. Моніторинг і контроль	Виміряти ефективність, відкоригувати при потребі	Підтверджена цінність зміни

Головні принципи:

✓ У PMBOK акцент на формалізації (Change Requests, CCB).

✓ В Agile акцент на швидкості та адаптивності (беклог, спринти).

✓ В обох підходах важливе: прозорість, фіксація результатів, зворотний зв'язок.

Висновок. Якісне управління змінами дозволяє:

✓ уникати хаосу в проєктах;

✓ підтримувати баланс між стабільністю та гнучкістю;

✓ досягати бізнес-цілей без втрат у якості продукту.

7.3.3. Ролі: Project Manager, Business Analyst, Stakeholders

Управління змінами – це командна діяльність, де різні ролі мають власні обов'язки і зони відповідальності (табл.7.15)

1. Project Manager (PM).

✓ Координує процес змін, забезпечує прозорість та дотримання методології.

✓ Веде комунікацію з усіма учасниками процесу.

✓ Забезпечує баланс між бюджетом, часом і якістю.

✓ Відповідає за оновлення плану проєкту після затвердження змін.

2. Business Analyst (BA).

✓ Фіксує та формалізує потреби у змінах (Change Request, User Story).

✓ Оцінює вплив змін на бізнес-процеси, користувачів і функціональність продукту.

✓ Допомагає PM готувати матеріали для ухвалення рішень.

✓ Є «посередником» між замовником (бізнесом) і технічною командою.

3. Stakeholders (зацікавлені сторони).

✓ Замовники, інвестори, кінцеві користувачі, топ-менеджмент.

✓ Вони формують потребу у змінах і визначають бізнес-цінність.

✓ Мають право ініціювати зміни та брати участь у прийнятті ключових рішень.

✓ В Agile – Stakeholders часто взаємодіють із Product Owner через беклог.

Таблиця 7.15 – Ролі у процесі управління змінами

Роль	Основні завдання	Приклад відповідальності
Project Manager (PM)	Координація процесу змін, баланс між часом, бюджетом і якістю, оновлення плану проєкту.	Організація засідання Change Control Board, оновлення графіку релізів.
Business Analyst (BA)	Фіксація та формалізація вимог, аналіз впливу на бізнес-процеси, допомога у прийнятті рішень.	Оформлення Change Request, підготовка сценаріїв впливу на користувачів.
Stakeholders	Формування потреб у змінах, визначення бізнес-цінності, ініціація змін.	Маркетинговий відділ подає запит на нову функцію; інвестори вимагають відповідності нормативам.

Приклад із практики.

У корпоративному проєкті зміна ініціюється відділом маркетингу (Stakeholders).

Рішення: BA оформлює бізнес-вимоги, описує очікуваний результат. PM оцінює вплив зміни на строки та бюджет і представляє план впровадження керівництву.

Висновок. Успішне управління змінами можливе лише тоді, коли ролі чітко визначені, а комунікація між ними налагоджена.

7.4. Методи впровадження змін

Моделі впровадження змін спирається на команду проєкту, процеси, ресурси і будь-яка модель визначається з цими складовими.

7.4.1. ADKAR-модель (Awareness, Desire, Knowledge, Ability, Reinforcement)

ADKAR – це одна з найпоширеніших моделей управління змінами, розроблена консалтинговою компанією Prosci. Вона фокусується не лише на процесах, а насамперед на людях, які беруть участь у змінах.

Модель складається з 5 послідовних елементів:

1. Awareness (усвідомлення).

✓ Люди повинні розуміти, чому зміни потрібні.

✓ Якщо команда або користувачі не розуміють причин, виникає опір.

✓ Приклади інструментів: комунікаційні кампанії, зустрічі з командою, презентації.

2. Desire (бажання).

✓ Учасники повинні хотіти підтримати зміни.

✓ Це формується через мотивацію, винагороди, демонстрацію вигоди.

✓ Приклад: показати, як нова система полегшить роботу співробітників.

3. Knowledge (знання).

✓ Необхідно навчити людей, як саме реалізувати зміни.

✓ Використовуються тренінги, інструкції, воркшопи.

✓ Приклад: навчання співробітників роботі з новою CRM.

4. Ability (здатність).

✓ Працівники повинні мати навички і ресурси, щоб застосувати знання на практиці.

✓ Приклад: тестове середовище, підтримка менторів, доступ до інструментів.

5. Reinforcement (підкріплення).

✓ Зміни потрібно закріпити, щоб вони стали частиною культури.

✓ Це робиться через контроль, регулярні рев'ю, винагороди, зворотний зв'язок.

✓ Приклад: бонуси за використання нових процесів, включення змін до KPI.

Сильні сторони ADKAR:

✓ Орієнтація на людей, а не лише на процес.

✓ Добре працює в IT-командах, де потрібне прийняття змін усіма учасниками.

✓ Підходить для впровадження нових технологій, методологій або інструментів.

Приклади етапів ADKAR для IT-сфери наведені у таблиці 7.16.

Таблиця 7.16 – Етапи ADKAR та приклади з ІТ

Етап	Що означає	Приклад в ІТ-проєкті
Awareness (усвідомлення)	Розуміння причин змін	Комунікація з командою про перехід на Agile для швидших релізів
Desire (бажання)	Мотивація підтримати зміни	Показати вигоди Scrum: більше впливу на продукт, прозорість
Knowledge (знання)	Навчання новим процесам чи технологіям	Тренінги з Jira та Scrum для розробників і тестувальників
Ability (здатність)	Практичні навички впровадження змін	Пілотні спринти з підтримкою менторів
Reinforcement (підкріплення)	Закріплення змін у культурі команди	Регулярні ретроспективи, KPI з Agile-практик

Приклад застосування.

Компанія впроваджує Agile-методологію замість Waterfall.

- ✓ Awareness: пояснюють, що це підвищить швидкість релізів.
- ✓ Desire: мотивують команду можливістю впливати на продукт.
- ✓ Knowledge: проводять тренінги з Scrum.
- ✓ Ability: запускають кілька пілотних спринтів.
- ✓ Reinforcement: закріплюють практику через ретроспективи та внутрішні KPI.

Висновок. ADKAR-модель допомагає врахувати людський фактор, який є ключовим для успішних змін у проєктах.

7.4.2. Модель Джона Коттера (Kotter’s 8 Steps)

Модель, розроблена професором Гарвардської бізнес-школи Джоном Коттером, описує послідовність із 8 кроків, які допомагають організаціям успішно реалізовувати зміни. Вона орієнтована на лідерство і культуру організації.

8 кроків Коттера (табд.7.17):

1. Створення відчуття терміновості (Create Urgency). Пояснити команді, чому зміни необхідні зараз.

Приклад: показати дані про відставання продукту від конкурентів.

2. Формування потужної коаліції (Build a Guiding Coalition). Залучити ключових лідерів і впливових осіб, які підтримуватимуть зміни.

Приклад: сформувати групу з PM, BA, технічного лідера та бізнес-спонсора.

3. Створення бачення і стратегії (Form a Vision and Strategy). Чітко сформулювати, куди ведуть зміни і яку цінність вони принесуть.

Приклад: «Перехід на хмарні сервіси зменшить витрати на 30 %».

4. Комунікація бачення (Communicate the Vision). Постійно пояснювати і підтримувати ідею змін.

Приклад: регулярні збори, презентації, FAQ.

5. Надання можливостей для дій (Empower Action). Усунути бар'єри, які заважають реалізації змін.

Приклад: надати доступ до нових інструментів, навчання, ресурси.

6. Швидкі перемоги (Generate Short-Term Wins). Показати перші результати, щоб підвищити мотивацію.

Приклад: пілотний проєкт з впровадження Agile у невеликій команді.

7. Консолідація досягнень і продовження змін (Consolidate Gains and Produce More Change). Використати успіхи як основу для наступних змін.

Приклад: після успішного пілоту поширити Agile на всю компанію.

8. Закріплення змін у культурі (Anchor New Approaches in the Culture). Зробити нові підходи частиною корпоративної культури.

Приклад: включення Agile-метрик у KPI команди.

Приклад застосування.

У компанії вирішили перейти з локальних серверів на хмарну інфраструктуру:

✓ Створено відчуття терміновості через показ ризиків застарілої системи.

✓ Сформовано коаліцію з IT-директора, PM і DevOps.

✓ Сформульовано бачення – «хмара зменшить витрати і підвищить безпеку».

✓ Перший успіх – запуск тестового середовища в хмарі.

✓ Згодом зміна поширилася на всі відділи.

Таблиця 7.17 – 8 кроків моделі Коттера

Крок	Суть	Приклад в ІТ
1. Створення відчуття терміновості	Пояснити, чому зміни потрібні зараз	Дані про відставання продукту від конкурентів
2. Формування коаліції	Залучити ключових лідерів для підтримки	Команда з PM, BA, техліда та бізнес-спонсора
3. Створення бачення і стратегії	Визначити цінність змін і стратегічні цілі	«Перехід у хмару зменшить витрати на 30 %»
4. Комунікація бачення	Регулярно пояснювати і підтримувати ідею	Презентації, зустрічі, FAQ
5. Надання можливостей для дій	Прибрати бар'єри для реалізації змін	Доступ до нових інструментів, навчання
6. Швидкі перемоги	Показати перші результати для мотивації	Пілотний Agile-проєкт у невеликій команді
7. Консолідація досягнень	Використати успіх для подальших змін	Поширення Agile на всі відділи компанії
8. Закріплення змін у культурі	Інтегрувати зміни у корпоративну культуру	Agile-метрики у KPI команди

Висновок. Модель Коттера допомагає структурувати процес змін, забезпечити підтримку лідерів і закріпити нові практики у культурі компанії.

7.4.3. Порівняння структурованих і гнучких методів

Управління змінами можна реалізовувати за двома основними підходами:

- ✓ Структуровані методи (PMBOK, PRINCE2, Коттер, ADKAR).
- ✓ Гнучкі методи (Agile, Scrum, Kanban, Lean Change Management).

1. Структуровані методи:

- ✓ Побудовані на чітких кроках і формальних процесах.
- ✓ Використовують документацію (Change Request, CCB, матриці відповідальності).

✓ Підходять для великих і регульованих галузей (державні проєкти, банківська сфера, оборонка).

Приклад: у впровадженні банківської IT-системи зміни погоджуються через комітет, із суворим контролем.

2. Гнучкі методи:

✓ Орієнтовані на адаптивність і швидку реакцію.

✓ Зміни інтегруються без складної бюрократії – через беклог і пріоритезацію.

✓ Підходять для стартапів і продуктових компаній, де важлива швидкість і гнучкість.

Порівняння критеріїв структурованих та гнучких методів наведені у таблиці 7.18.

Таблиця 7.18 – Порівняння структурованих та гнучких методів

Критерій	Структуровані методи	Гнучкі методи
Фокус	Процеси, документація	Люди, результат
Швидкість реакції	Низька	Висока
Формалізація	Висока (регламенти, комітети)	Низька (беклог, воркшопи)
Сфера застосування	Великі корпоративні і держ. проєкти	Стартапи, продуктові IT-команди
Приклад	Впровадження ERP у банку	Додавання нової функції у мобільний застосунок

Приклад: у Scrum будь-яка зміна вноситься в Product Backlog і може потрапити в наступний спринт.

Висновок. Структуровані методи забезпечують контроль і передбачуваність, але втрачають у швидкості. Гнучкі методи дозволяють швидко реагувати на зміни, але потребують зрілої команди й довіри. У сучасній практиці часто застосовують гібридні моделі: стратегічні зміни ведуться структуровано, а операційні – гнучко.

7.4.4. Приклади застосування у корпоративних та стартап-проєктах

1. Корпоративні проєкти.

У великих компаніях і державних організаціях зміни зазвичай масштабні, з великою кількістю стейкхолдерів і високим рівнем формалізації.

Методи: частіше використовують структуровані моделі (PMBOK Change Management, Коттер, ADKAR).

Приклад:

✓ У банку впроваджують нову CRM-систему.

✓ BA фіксує вимоги, PM створює Change Request, рішення ухвалює Change Control Board.

✓ Використовується модель Коттера: створюється бачення змін, формуються швидкі перемоги (наприклад, пілот у відділі продажів), а потім поширення на всю організацію.

✓ Результат: контрольований процес, низькі ризики, відповідність законодавчим нормам.

2. Стартапи

У стартапах важлива швидкість і адаптивність, тому структуровані процеси можуть гальмувати розвиток.

Методи: переважають гнучкі моделі (Agile Change Management, Lean Change, Scrum, Kanban).

Приклад:

✓ Стартап створює мобільний додаток для доставки.

✓ Після запуску MVP отримують фідбек: користувачі хочуть інтеграцію з Apple Pay.

✓ Зміна вноситься в беклог, команда планує інтеграцію вже в наступному спринті.

✓ ADKAR використовується для внутрішньої роботи з командою (Awareness – чому це критично; Knowledge – як інтегрувати; Reinforcement – відстеження результату).

✓ Результат: швидке реагування на потреби користувачів, мінімальна бюрократія, висока швидкість релізів.

Порівняння:

✓ У корпорації – головне стабільність, передбачуваність, відповідність стандартам.

✓ У стартапі – головне гнучкість, швидкість, адаптивність.

Часто в реальності застосовується гібридний підхід: стратегічні зміни – структуровано, операційні – Agile.

У таблиці 7.19 наведені приклади застосування методів.

Таблиця 7.19 – Приклади застосування

Тип проєкту	Метод	Приклад	Результат
Корпоративний	PMBOK, Коттер, ADKAR	Впровадження нової CRM у банку. BA фіксує вимоги, PM формує Change Request, рішення ухвалює CCB.	Контрольований процес, низькі ризики, відповідність законодавчим нормам.
Стартап	Agile, Scrum, Lean Change	MVP додатка для доставки. Отримано фідбек: інтеграція Apple Pay. Зміна вноситься в беклог і реалізується у спринті.	Швидке реагування на потреби користувачів, мінімальна бюрократія, швидкі релізи.

Висновок. У корпораціях головне – стабільність, передбачуваність, відповідність стандартам. У стартапах – гнучкість, швидкість, адаптивність. Часто застосовується гібридний підхід: стратегічні зміни ведуться структуровано, а операційні – за Agile.

7.5. Інструменти відстеження змін

Відстеження змін, як і будь-який творчий процес, повинен бути контрольованим і для цього існують певний інструментарій

7.5.1. Jira (change requests, backlog refinement)

Jira – це одна з найпопулярніших систем для управління завданнями та проєктами, яку активно використовують в IT-командах для контролю змін.

Можливості Jira у контексті управління змінами (табл.7.20):

1. Change Requests (запити на зміни).

У Jira можна створювати окремий тип задачі Change Request. Він дозволяє формалізувати ініціативу зміни: опис проблеми, очікуваний результат, оцінку впливу. PM або BA додають деталі, команда оцінює трудомісткість, а стейкхолдери ухвалюють рішення.

Приклад: «Потрібно інтегрувати новий платіжний сервіс».

2. Backlog Refinement (уточнення беклогу).

У гнучких командах зміни часто потрапляють у Product Backlog. Jira дозволяє пріоритезувати зміни, планувати їх у спринти та відстежувати статус.

Приклад: фідбек користувачів додається у backlog, під час refinement він перетворюється на user story.

3. Прозорість і відстежуваність.

Будь-яка зміна має статус («Open», «In Progress», «Done»), виконавця та дедлайн. Це дозволяє всім учасникам проєкту бачити актуальний стан змін.

4. Інтеграція з іншими інструментами.

Jira інтегрується з Confluence (для документації), Bitbucket/GitHub (для коду), Slack/Teams (для сповіщень). Це створює єдину екосистему від ідеї зміни до її реалізації.

Таблиця 7.20 – Використання Jira для управління змінами

Функція	Опис	Приклад
Change Requests	Окремий тип задачі для фіксації змін: опис проблеми, очікуваний результат, оцінка впливу.	Запит на інтеграцію нового платіжного сервісу.
Backlog Refinement	Зміни потрапляють у backlog, уточнюються та пріоритезуються для наступних спринтів.	Фідбек користувачів → user story у backlog.
Прозорість і відстежуваність	Статуси («Open», «In Progress», «Done»), виконавець і дедлайн відображаються у системі.	Усі стейкхолдери бачать актуальний стан змін.
Інтеграції	Синхронізація з Confluence, Bitbucket, Slack/Teams.	Автоматичне створення задач із коментарів у Slack.

Приклад із практики. У стартапі після релізу з'явилася потреба додати темну тему.

✓ BA створив у Jira Change Request.

✓ Продуктовий власник переніс його у backlog.

✓ На refinement сесії команда розбила задачу на кілька user stories («зміна кольорів UI», «тестування accessibility»).

✓ Задача була виконана у наступному спринті й зафіксована у release notes.

Висновок. Jira дозволяє організувати управління змінами прозоро й ефективно: від фіксації запиту до його реалізації та відстеження результатів у спільній системі.

7.5.2. Confluence (документація змін, історія версій)

Confluence – це корпоративний wiki-сервіс від Atlassian, який використовується для зберігання документації, колективної роботи та управління знаннями. У контексті управління змінами він відіграє ключову роль у фіксації та відстеженні всієї документації, пов'язаної зі змінами.

Можливості Confluence для управління змінами (табл.7.21):

1. Документація змін.

У Confluence створюють сторінки з описом Change Requests, технічних рішень, протоколів засідань. Це дає змогу всім учасникам мати доступ до єдиного джерела правди.

Приклад: окрема сторінка «Інтеграція з Apple Pay» із описом вимог і планом реалізації.

2. Історія версій.

Кожна сторінка має повну історію змін – хто і коли редагував документ. Це забезпечує прозорість і контроль у випадку конфліктів.

Приклад: перегляд, які зміни вносив BA у вимоги протягом тижня.

3. Шаблони для управління змінами.

Confluence пропонує готові шаблони: Change Request, Decision Log, Meeting Notes. Це прискорює створення документації.

4. Інтеграція з Jira.

Завдання в Jira можна пов'язати з документацією у Confluence.

Приклад: user story у Jira посилається на сторінку з описом технічного рішення в Confluence.

Таблиця 7.21 – Використання Confluence для управління змінами

Функція	Опис	Приклад
Документація змін	Створення сторінок із описом Change Requests, рішень, протоколів.	Сторінка «Інтеграція з Apple Pay» з вимогами й планом реалізації.
Історія версій	Збереження всіх змін документа з авторством і датами.	Перегляд змін у вимогах, які вносив BA протягом тижня.
Шаблони	Готові шаблони для Change Request, Decision Log, Meeting Notes.	Використання шаблону для протоколу зустрічі з клієнтом.
Інтеграція з Jira	Пов’язування Jira-тикетів із документацією в Confluence.	User story у Jira посилається на сторінку з описом технічного рішення.

Приклад із практики.

У корпорації під час впровадження ERP-системи:

✓ Усі Change Requests документуються у Confluence.

✓ PM додає коментарі й посилання на Jira-тикети.

✓ BA редагує вимоги, а стейкхолдери переглядають зміни через історію версій.

✓ Це дозволяє уникати непорозумінь і мати прозорий журнал змін.

Висновок. Confluence забезпечує єдине джерело правди для всієї документації змін, дозволяючи командам зберігати прозорість та ефективно співпрацювати завдяки історії версій, шаблонам і інтеграції з Jira.

7.5.3. IBM DOORS (Dynamic Object Oriented Requirements System)

IBM DOORS – це потужний інструмент для управління вимогами та змінами, який використовується у великих корпораціях, зокрема в авіації, оборонній промисловості, телекомі, енергетиці.

Можливості IBM DOORS у контексті управління змінами (табл.7.22):

1. Трасування вимог.

Дозволяє пов’язати бізнес-вимоги з функціональними, технічними специфікаціями та тест-кейсами. При зміні однієї вимоги система показує, які інші артефакти будуть зачеплені.

2. Контроль версій.

DOORS зберігає історію всіх змін вимог. Можна порівняти будь-які дві версії документа та побачити різницю.

3. Робота з великими проєктами.

Підтримує тисячі вимог у багаторівневій структурі. Зручно для складних систем (наприклад, авіоніка чи телекомунікаційна інфраструктура).

4. Інтеграція.

DOORS інтегрується з інструментами тестування (IBM Rational Quality Manager), управління проєктами (MS Project) та DevOps-платформами.

Таблиця 7.22 – Використання IBM DOORS для управління змінами

Функція	Опис	Приклад
Трасування вимог	Зв’язування бізнес-вимог із технічними специфікаціями та тест-кейсами.	При зміні однієї вимоги система показує пов’язані артефакти.
Контроль версій	Збереження історії змін вимог, можливість порівняти різні версії.	ВА бачить, як змінювалися функціональні вимоги протягом року.
Робота з великими проєктами	Підтримка тисяч вимог у багаторівневій структурі.	В авіоніці відслідковуються понад 20 000 вимог.
Інтеграція	Підключення до інструментів тестування та управління проєктами.	Синхронізація з IBM Rational Quality Manager для перевірки тест-кейсів.

Приклад із практики.

В оборонному проєкті компанія розробляє нову систему управління

зв’язком.

✓ У DOORS внесено понад 15 000 вимог.

✓ При зміні в специфікації однієї функції система автоматично показала, які тест-кейси та технічні модулі потрібно оновити.

✓ Це дозволило уникнути «ефекту доміно» та зекономити місяці роботи.

Висновок. IBM DOORS – це інструмент, орієнтований на великі та складні проєкти, де управління змінами потребує високої точності, масштабованості та інтеграції з іншими системами. Він дозволяє зменшити ризики та зберегти контроль над усім життєвим циклом вимог.

7.5.4. Git/GitHub/GitLab (версійність коду)

Git – це розподілена система контролю версій, яка використовується для відстеження змін у програмному коді.

GitHub і GitLab – це платформи, що базуються на Git і додають функціонал для командної співпраці, автоматизації та управління проєктами.

Можливості Git/GitHub/GitLab для управління змінами (табл.7.23):

1. Контроль версій коду.

Кожна зміна зберігається у вигляді коміту. Це дозволяє повернутися до будь-якої версії коду та аналізувати історію змін.

Приклад: відкат до попередньої версії після виявлення критичного бага.

2. Гілкування (branching).

Кожна нова зміна (фіча, багфікс) розробляється у власній гілці. Це дозволяє працювати паралельно над різними змінами без конфліктів.

Приклад: створення гілки feature/payment-integration.

3. Pull/Merge Requests.

Перед тим як інтегрувати зміни у головну гілку, вони проходять код-рев’ю. Це забезпечує контроль якості та колективне ухвалення змін.

Приклад: зміни перевіряються колегами через GitHub Pull Request.

4. Issue tracking.

GitHub/GitLab дозволяють прив’язати зміни в коді до задач (issues). Це створює прозорість: яка вимога/баг реалізується яким комітом.

5. CI/CD інтеграція.

Після внесення змін можуть автоматично запускатися тести та деплой. Це знижує ризик помилок і пришвидшує впровадження змін.

Таблиця 7.23 – Використання Git/GitHub/GitLab для управління змінами

Функція	Опис	Приклад
Контроль версій коду	Кожна зміна зберігається як коміт, що дозволяє відстежувати історію та робити відкат.	Відкат до попередньої версії після критичного бага.
Гілкування (branching)	Зміни розробляються у власних гілках, що дозволяє працювати паралельно.	Гілка feature/payment-integration для нової функції.
Pull/Merge Requests	Перед злиттям змін у головну гілку проводиться код-рев'ю.	Pull Request у GitHub для перевірки змін колегами.
Issue tracking	Зміни в коді можна прив'язати до задач або багів у системі.	Issue «#45 Додати push-сповіщення» пов'язаний із комітами.
CI/CD інтеграція	Автоматичний запуск тестів і деплой після змін у репозиторії.	Зміна в main запускає збірку та деплой на тестове середовище.

Приклад із практики.

У продуктовій компанії команда працює над мобільним застосунком:

✓ BA створює задачу «додати push-сповіщення».

✓ Dev створює гілку feature/push-notifications.

✓ Після завершення роботи створюється Pull Request у GitHub.

✓ Код перевіряється іншими розробниками, запускаються автоматичні тести.

✓ Після злиття зміна автоматично деплоїться на тестове середовище.

Висновок. Git разом із GitHub/GitLab забезпечує надійний контроль версій коду, прозорість змін і автоматизацію процесів, що робить його незамінним інструментом управління змінами в IT-проєктах.

7.5.5. Change Request Forms

Change Request Form (CRF) – це формалізований документ, який використовується для ініціації, опису та оцінки змін у проєкті. Він особливо поширений у традиційних підходах (PMBOK, PRINCE2), але може використовуватися і в гнучких середовищах, якщо потрібно фіксувати важливі зміни.

Основні елементи Change Request Form (табл.7.24):

1. Ідентифікатор (ID). Унікальний номер зміни.
2. Дата та ініціатор. Хто і коли подав запит.
3. Опис зміни. Яку саме функцію, процес чи вимогу потрібно змінити.
4. Причина зміни. Чому вона необхідна (нові вимоги бізнесу, баг, регуляторні норми).
5. Аналіз впливу. Як зміна вплине на обсяг робіт, бюджет, строки, якість.
6. Пропоноване рішення. Варіанти впровадження.
7. Статус. (Open, Under Review, Approved, Rejected, Deferred).
8. Підписи/затвердження. Хто схвалив зміну (PM, Steering Committee, Stakeholders).

Таблиця 7.24 – Шаблон Change Request Form

ID зміни	
Дата та ініціатор	
Опис зміни	
Причина зміни	
Аналіз впливу	
Пропоноване рішення	
Статус (Open/Approved/Rejected)	
Підписи/затвердження	
Коментарі	

Переваги використання CRF:

- ✓ Прозорість і формалізація змін.
- ✓ Легко відстежувати історію рішень.

✓ Важливо для юридичної та контрактної відповідності.

Приклад із практики.

У державному IT-проєкті було отримано запит на інтеграцію з новим державним реєстром.

✓ BA оформив Change Request Form із описом потреби.

✓ PM провів аналіз впливу на бюджет і строки.

✓ Steering Committee затвердив зміну, і вона була додана до проєктного плану.

Висновок. Change Request Forms забезпечують прозорість і формалізацію процесу управління змінами. Вони дозволяють відстежувати історію рішень і зберігати юридичну відповідність у складних або контрактних проєктах.

7.5.6. Traceability у матриці вимог (RTM)

Матриця трасування вимог (RTM) – це таблиця, яка відображає зв'язки між бізнес-вимогами, функціональними вимогами, технічними специфікаціями, тест-кейсами та результатами тестування. Її головна мета – забезпечити цілісність проєкту та прозорість впливу змін.

Ключові функції RTM для управління змінами (табл.7.25):

1. Відстеження впливу змін.

Якщо змінюється бізнес-вимога, RTM показує, які функціональні модулі та тести потребують оновлення.

2. Повнота реалізації.

Дає змогу перевірити, чи всі вимоги мають реалізацію в коді та тестах.

3. Прозорість для стейкхолдерів.

PM та BA можуть показати клієнту, які саме вимоги були реалізовані й протестовані.

Приклад із практики.

У фінансовому проєкті клієнт вніс зміни у вимогу «Підтримка Apple Pay».

✓ У RTM було видно, що зміни зачіпають 2 функціональні вимоги, 3 модулі коду та 4 тест-кейси.

✓ Завдяки цьому команда уникла пропуску критичних оновлень і

швидко внесла зміни у всі пов’язані артефакти.

Таблиця 7.25 – Приклад структури RTM

ID вимоги	Бізнес-вимога	Функціональна вимога	Технічна реалізація	Тест-кейс	Статус
BR-01	Система повинна дозволяти онлайн-оплати	Реалізувати інтеграцію з платіжним шлюзом	API PaymentService	TC-05 Перевірка оплати карткою	Done
BR-02	Користувач повинен мати можливість змінювати пароль	Додати форму «Change Password»	UserService + DB Update	TC-10 Зміна паролю	In Progress

Висновок. RTM є ключовим інструментом для відстеження змін у складних проєктах. Він допомагає контролювати повноту реалізації вимог, знижує ризик помилок та забезпечує прозорість для всіх стейкхолдерів.

7.5.7. Порівняння інструментів управління змінами

У таблиці 7.26 наведено порівняння основних інструментів управління змінами, які застосовуються у різних типах ІТ-проєктів. Кожен інструмент має свої переваги, сферу застосування та рівень формалізації.

Висновок. Кожен інструмент має свою нішу: Jira та Confluence зручні для Agile-команд, IBM DOORS – для складних корпоративних систем, GitHub/GitLab – для управління змінами у коді, Change Request Forms – для традиційних і державних проєктів, RTM – для контролю цілісності вимог. На практиці часто використовується комбінація кількох інструментів залежно від потреб.

Таблиця 7.26 – Порівняння інструментів

Інструмент	Призначення	Сильні сторони	Обмеження	Приклад застосування
Jira	Управління задачами, беклогом, change requests	Прозорість, інтеграція з Confluence/CI/CD, підтримка Agile	Може бути перевантажена великою кількістю тикетів	Agile-команди стартапу – backlog refinement для нових фіч
Confluence	Документація змін, історія версій, knowledge base	Єдине джерело правди, шаблони, інтеграція з Jira	Менш зручний для відстеження дрібних завдань	Корпоративний проєкт – ведення Change Requests та протоколів зустрічей
IBM DOORS	Трасування вимог у великих системах	Масштабність, контроль версій, інтеграція з тестуванням	Висока вартість, складність у використанні	Оборонна промисловість – 15 000+ вимог з трасуванням до тестів
Git/GitHub/ GitLab	Контроль версій коду та CI/CD	Гілкування, код-рев'ю, автоматизація деплою	Зосередженість лише на коді (не охоплює бізнес-вимоги)	Продуктова компанія – pull requests та issue tracking
Change Request Forms	Формалізовані запити на зміни	Прозорість, юридична відповідність, контроль	Бюрократія, повільність у порівнянні з Agile	Державний проєкт – інтеграція з реєстрами
RTM (матриця трасування)	Зв'язки між вимогами, тестами та реалізацією	Контроль цілісності, повноти реалізації, прозорість	Складність підтримки в актуальному стані	Фінансовий проєкт – інтеграція Apple Pay із 3 модулями та 4 тест-кейсами

7.6. Загальні висновки

7.6.1. Основні тези лекції

1. Зміни – невід'ємна частина IT-проєктів. Вони виникають через розвиток бізнесу, ринкові фактори, зовнішні умови чи технологічні

обмеження.

2. Процеси управління змінами включають ідентифікацію потреби, аналіз впливу, ухвалення рішень, реалізацію та контроль результатів.

3. Методології впровадження змін (ADKAR, Kotter) допомагають структурувати роботу з людьми та процесами.

4. Інструменти управління змінами (Jira, Confluence, DOORS, Git/GitHub/GitLab, Change Request Forms, RTM) забезпечують прозорість, контроль і зменшення ризиків.

5. Вибір інструментів залежить від типу проєкту:

✓ стартапи використовують легкі й гнучкі підходи (Jira, GitHub),

✓ корпоративні проєкти застосовують більш формалізовані рішення (Confluence, DOORS, CRF, RTM).

6. Оптимальним підходом часто є комбінація інструментів, що дозволяє балансувати між гнучкістю та контролем.

7.6.2. Питання для самоперевірки

1. Чому зміни є природною частиною ІТ-проєктів?
2. Які основні етапи процесу управління змінами ви знаєте?
3. У чому відмінність підходів ADKAR та Kotter?
4. Коли доцільно застосовувати Change Request Forms?
5. Для чого використовується RTM і як вона допомагає при змінах?
6. Які можливості Jira забезпечують відстеження змін?
7. Як Confluence допомагає документувати та контролювати зміни?
8. Чому IBM DOORS актуальний для великих корпоративних і державних систем?
9. Які переваги Git/GitHub/GitLab у відстеженні змін у коді?
10. Наведіть приклад, коли гібридне використання інструментів є найефективнішим.

7.7. Контрольні запитання

1. Чому зміни є невід'ємною частиною будь-якого ІТ-проєкту? Назвіть основні джерела змін.

2. Яка «вартість змін» у контексті ІТ-проєкту? Опишіть фінансову,

часову, людську та технічну складові.

3. Що таке неконтрольовані зміни (scope creep) і які їх наслідки для проєкту?

4. Що таке процес управління змінами (change management process)? Опишіть основні кроки.

5. Що таке запит на зміну (change request) і яку роль він відіграє у формальному управлінні змінами?

6. Як працює комітет з контролю змін (Change Control Board, CCB)? Яка його роль у прийнятті рішень?

7. Як управління змінами реалізоване у методології Waterfall порівняно з Agile?

8. Чому кожна зміна вимог може породжувати нові ризики? Поясніть взаємозв'язок між вимогами, змінами та ризиками.

9. Які регуляторні вимоги (наприклад, GDPR) можуть ініціювати зміни в архітектурі системи?

10. Як зворотний зв'язок від користувачів під час бета-тестування впливає на управління змінами?

11. Що таке версійний контроль і як він пов'язаний з управлінням змінами у розробці ПЗ?

12. Яку роль відіграє бізнес-аналітик в управлінні змінами? Опишіть конкретні дії BA при надходженні запиту на зміну.

13. Як оцінити вплив запропонованої зміни на трикутник обмежень (час, вартість, якість)?

14. Що таке технічний борг і як він пов'язаний із неконтрольованими змінами?

15. Наведіть приклад успішного або неуспішного управління змінами в IT-проєкті. Що можна було зробити краще?

8. ІНСТРУМЕНТИ ТА ПРАКТИКИ СУЧАСНОГО УПРАВЛІННЯ ІТ-ПРОЄКТАМИ

8.1. Вступ

Ця тема присвячена узагальненому використанню інструментів для управління проєктами.

8.1.1. Чому сучасні інструменти важливі для управління ІТ-проєктами

В сучасних умовах ІТ-проєкти відзначаються високою складністю, динамічністю та необхідністю швидкого реагування на зміни. Команди працюють у розподіленому середовищі, часто залучаючи спеціалістів із різних країн та часових зон. У таких умовах класичні методи управління (Excel-таблиці, паперова документація) стають недостатніми.

1. Сучасні інструменти управління проєктами виконують кілька ключових функцій:

2. Прозорість процесів. Учасники бачать поточний стан завдань, дедлайни, відповідальних.

3. Співпраця в реальному часі. Інструменти дозволяють команді працювати синхронно, навіть якщо всі члени розподілені географічно.

4. Відстеження змін і історії. Будь-яка зміна у вимогах, документації чи коді зберігається, що дає змогу уникати непорозумінь.

5. Автоматизація. Інтеграції з системами тестування, CI/CD, аналітикою зменшують рутинну роботу.

6. Аналітика та звітність. Інструменти формують метрики (velocity, lead time, burn-down charts), що допомагає PM та стейкхолдерам оцінювати прогрес.

Приклад із практики.

У стартапі без інструментів команда витрачає час на пошук актуальних завдань у месенджерах.

Після переходу на Trello чи Jira з’являється прозора дошка завдань, де видно прогрес, пріоритети та відповідальних. Це знижує хаос і прискорює доставку функціоналу.

Таблиця 8.1 – Проблеми → Рішення через сучасні інструменти

Проблема без інструментів	Рішення через сучасні інструменти
Завдання губляться у месенджерах	Jira/Trello – централізована дошка завдань
Важко зрозуміти прогрес проєкту	Burn-down charts, velocity у Jira
Немає історії змін	Confluence + Git – збереження версій
Ручна координація роботи	Автоматизація через CI/CD та інтеграції

8.1.2. Тенденції: автоматизація, візуалізація, аналітика

Сучасне управління IT-проєктами активно розвивається завдяки трьом ключовим тенденціям (табл.8.2):

1. Автоматизація

Інструменти управління дедалі більше інтегруються з системами DevOps, CI/CD, тестування. Автоматизовані нотифікації, звіти та інтеграція з календарями зменшують ручну роботу менеджера.

Приклад: при злитті коду в GitHub автоматично запускаються тести й створюється оновлений статус у Jira.

2. Візуалізація

Ключовим фактором є швидке розуміння стану проєкту всіма стейкхолдерами. Kanban-дошки, діаграми Ґанта, burn-down charts дозволяють оцінювати прогрес «з одного погляду».

Приклад: у Trello видно, які завдання в статусі “To Do”, “In Progress”, “Done”.

3. Аналітика

Інструменти все частіше мають вбудовану бізнес-аналітику: velocity, cycle time, delivery rate. Це дозволяє PM оцінювати ефективність команди та прогнозувати майбутні результати.

Таблиця 8.2 – Тренди сучасного управління IT-проєктами

Тенденція	Суть
Автоматизація	Інтеграція з CI/CD, тестуванням, зниження ручної роботи.
Візуалізація	Дошки, діаграми, графіки для прозорості прогресу.
Аналітика	Метрики продуктивності, прогнозування результатів.

Приклад: Jira автоматично рахує velocity команди за кілька спринтів, що дозволяє планувати релізи.

8.2. Інструменти управління проєктами

Підрозділ розглядає узагальнено інструменти для управління проєктами.

8.2.1. Jira: управління завданнями, беклогом, Agile-дошки

Jira – це один із найпопулярніших інструментів для управління ІТ-проєктами, створений компанією Atlassian. Він особливо поширений серед Agile-команд, які використовують Scrum чи Kanban.

Основні можливості Jira:

1. Управління завданнями.

✓ Створення задач (issues) різних типів: epic, story, task, bug.

✓ Призначення відповідальних, дедлайнів, пріоритетів.

✓ Відстеження статусів («To Do», «In Progress», «Done»).

2. Backlog Management (керування беклогом).

✓ Усі завдання зберігаються у product backlog.

✓ Product Owner може змінювати пріоритети, планувати спринти.

✓ Refinement-сесії допомагають деталізувати завдання перед розробкою.

3. Agile-дошки.

✓ Scrum board: робота зі спринтами, velocity, burn-down chart.

✓ Kanban board: безперервний потік завдань, контроль WIP (work in progress).

✓ Дошки можна налаштовувати під конкретний процес команди.

4. Звітність і аналітика.

✓ Jira генерує діаграми: burn-down chart, velocity chart, control chart. Це допомагає менеджерам бачити реальний прогрес і прогнозувати строки.

Приклад із практики:

У продуктовій компанії команда з 10 людей використовує Jira для розробки мобільного застосунку (табл.8.3).

✓ У backlog зібрано понад 200 user stories.

✓ На спринт відбирається 20 задач, які відображаються на Scrum-дошці.

✓ Щодня команда проводить stand-up, використовуючи дошку Jira як візуальний центр.

✓ PM готує звіти для керівництва, використовуючи burn-down chart.

Таблиця 8.3 – Потік роботи у Jira

Backlog	Sprint	Done
200+ user stories	Відібрано 20 задач у Scrum board	Завершені задачі зі звітом у burn-down chart

8.2.2. Trello: канбан-дошка для невеликих команд і стартапів

Trello – це простий та інтуїтивний інструмент управління завданнями, що базується на принципі канбан-дошки. Його часто обирають невеликі команди, стартапи чи компанії, яким потрібне швидке розгортання без складних налаштувань.

Основні можливості Trello:

1. Канбан-дошка.

✓ Завдання оформлюються у вигляді карток (cards).

✓ Колонки зазвичай представляють статуси («To Do», «In Progress», «Done»).

✓ Картки можна легко переміщати між колонками drag-and-drop.

2. Простота використання.

✓ Не потребує навчання – користувачі інтуїтивно розуміють логіку роботи.

✓ Мінімум бюрократії у створенні завдань.

3. Гнучкість.

✓ Кожна команда може налаштовувати свої колонки.

✓ Додаються теги, дедлайни, прикріплення файлів, чеклісти.

4. Інтеграції.

✓ Підтримує інтеграції з Google Drive, Slack, GitHub, Jira.

✓ Є можливість додавати автоматизацію через Butler (правила для карток).

Приклад із практики:

У стартапі з 5 осіб команда маркетологів і розробників використовує Trello:

✓ Колонки «Ideas», «To Do», «Doing», «Testing», «Done».

✓ Картки з завданнями містять чеклісти, файли, дедлайни.

✓ Завдяки простій структурі команда швидко розподіляє завдання і бачить прогрес.

Таблиця 8.4 – Канбан-дошка у Trello

To Do	In Progress	Done
Створити новий лендинг	Верстка головної сторінки	Тестування системи логіну
Підготувати рекламну кампанію	Налаштування Google Analytics	Публікація блогу

8.2.3. MS Project: планування, діаграми Ганта, критичний шлях

MS Project – це потужний інструмент управління проєктами від Microsoft, який використовується переважно у великих організаціях і проєктах з високим рівнем формалізації.

Основні можливості MS Project:

1. Планування завдань.

✓ Можливість визначати залежності між завданнями (початок-кінець, кінець-кінець тощо).

✓ Встановлення тривалості завдань, ресурсів і дедлайнів.

✓ Створення комплексних планів із сотень завдань.

2. Діаграма Ганта.

✓ Візуалізація всіх завдань у вигляді календарної шкали.

✓ Дає змогу бачити, які роботи виконуються паралельно, а які – послідовно.

✓ Допомагає відслідковувати затримки та їхній вплив на загальний графік.

3. Метод критичного шляху (CPM).

✓ Визначення ключових завдань, від яких залежить термін

завершення проєкту.

✓ Аналіз «вузьких місць», які можуть затримати весь проєкт.

✓ Дозволяє проєктному менеджеру правильно розподілити ресурси.

4. Управління ресурсами.

✓ Призначення людських і матеріальних ресурсів до завдань.

✓ Контроль завантаженості команди.

Приклад із практики:

У великій корпорації планується впровадження ERP-системи (табл.8.5).

✓ Проєкт триває 18 місяців і містить понад 500 завдань.

✓ MS Project використовується для побудови діаграми Ганта, що відображає всі етапи: аналіз, розробка, тестування, впровадження.

✓ CPM дозволяє визначити, що затримка у тестуванні модуля фінансів відкладе весь реліз.

✓ Завдяки MS Project керівництво бачить повну картину проєкту та може ухвалювати рішення про додаткові ресурси.

Таблийя 8.5 – Спрощена діаграма Ганта

Етап	Місяць 1-3	Місяць 4-6	Місяць 7-9	Місяць 10-12
Аналіз	■■■			
Розробка		■■■	■■■	
Тестування			■■■	■
Впровадження				■■■

8.2.4. Confluence: документація, база знань, інтеграція з Jira

Confluence – це інструмент для створення та зберігання документації, розроблений компанією Atlassian. Він часто використовується в парі з Jira, утворюючи екосистему для управління завданнями та знаннями.

Основні можливості Confluence:

1. Документація.

✓ Створення сторінок для специфікацій, протоколів зустрічей, планів проєктів.

✓ Можливість організовувати інформацію у просторах (spaces).

✓ Версійність: збереження історії змін.

2. База знань.

✓ Зберігання технічної та бізнес-інформації в централізованому місці.

✓ Пошук по документах, теги, структуровані шаблони.

✓ Використання як внутрішньої wiki для компанії.

3. Інтеграція з Jira.

✓ Автоматичне пов’язування завдань із документацією.

✓ Можливість бачити статус задач безпосередньо у документах.

✓ Створення Jira-завдань безпосередньо з Confluence.

4. Співпраця.

✓ Одночасне редагування документів кількома користувачами.

✓ Коментарі, згадки користувачів, швидке узгодження рішень.

Приклад із практики:

У міжнародній компанії команда створює внутрішній портал у Confluence:

✓ Зберігає архітектурні рішення, протоколи зустрічей, шаблони документації.

✓ Нові співробітники отримують доступ до бази знань і швидко адаптуються.

✓ Через інтеграцію з Jira будь-який документ можна пов’язати з епічними завданнями та бачити прогрес у режимі реального часу.

Порівняння Jira та Confluence приведена у таюлиці 8.6.

Таблиця 8.6 – Jira ↔ Confluence

Jira (управління завданнями)	Confluence (документація)
Завдання, беклог, Scrum/Kanban	Специфікації, протоколи, база знань
Статус задач, метрики	Документи з прив’язкою до задач
Автоматичні звіти	Wiki-коментарі, спільна робота

8.2.5. Порівняння інструментів

Порівняльна інформація стосовно інструментів (Jira, Trello, MS Project, Confluence) з критеріями: призначення, тип команд, ключові можливості, сильні та слабкі сторони, наведена у таблиці 8.7.

Таблиця 8.7 – Порівняльна таблиця інструментів управління IT-проєктами

Інструмент	Призначення	Тип команд	Ключові можливості	Сильні сторони	Слабкі сторони
Jira	Управління завданнями та Agile-процесами	Agile-команди (Scrum, Kanban), середні та великі компанії	Backlog, Scrum/Kanban boards, звітність (burn-down, velocity)	Гнучкість налаштувань, інтеграція з DevOps, потужна аналітика	Складність для новачків, потребує налаштувань
Trello	Візуалізація завдань у вигляді канбан-дошки	Невеликі команди, стартапи, креативні проєкти	Картки, чеклісти, теги, дедлайни, інтеграції	Простота, швидке освоєння, інтуїтивний інтерфейс	Обмежені аналітичні можливості, не підходить для великих проєктів
MS Project	Планування великих і складних проєктів	Корпорації, державні та довготривалі проєкти	Діаграми Ганта, CPM, управління ресурсами	Потужне планування, контроль ресурсів, прогнозування	Висока складність, потреба у навчанні, важкий для Agile
Confluence	Документація та база знань	Будь-які команди, особливо в комбінації з Jira	Wiki-сторінки, шаблони, інтеграція з Jira	Централізована база знань, історія змін, спільна робота	Без Jira менш корисний, інтерфейс складніший за Google Docs

8.2.6. Приклади використання в різних типах компаній

1. Стартапи

Використовують Trello або легку конфігурацію Jira. Основний акцент – швидкість і гнучкість, а не формальність.

Приклад: команда з 6 людей керує беклогом у Trello з колонками «Ideas → To Do → Doing → Done». Завдання швидко змінюються залежно від

потреб інвесторів і користувачів.

2. Корпорації

Найчастіше комбінують Jira + Confluence. Jira використовується для управління завданнями (Scrum/Kanban), а Confluence – як централізована база знань.

Приклад: міжнародна компанія з 200 розробниками працює в кількох потоках (streams), кожен має свій backlog у Jira, але документація стандартизована в Confluence. Це дає прозорість і масштабованість.

3. Державні проєкти / великі інтеграції

Найбільше підходить MS Project, бо вимагається висока формалізація та контроль строків і ресурсів.

Приклад: державна програма цифровізації освіти на 2 роки з чітко визначеними етапами (аналіз, закупівлі, впровадження). Використовується діаграма Ганта, критичний шлях (CPM) і регулярна звітність у форматі, зрозумілому для державних структур.

8.3. Використання діаграм для управління та аналізу

Підрозділ розглядає інструменти роботи з діаграмами для управління проєктами.

8.3.1. BPMN (Business Process Model and Notation) – моделювання бізнес-процесів

BPMN – це міжнародний стандарт (розроблений Object Management Group), який використовується для графічного відображення бізнес-процесів. Він дозволяє представити складні бізнес-операції у вигляді зрозумілих блок-схем, що однаково легко читаються як технічними фахівцями, так і менеджментом.

Основні елементи BPMN:

1. Події (Events) – позначають початок, завершення чи проміжні події процесу.

- ✓ Початкова подія (Start Event).
- ✓ Кінцева подія (End Event).
- ✓ Проміжна подія (Intermediate Event).

2. Дії/Задачі (Activities/Tasks) – конкретні кроки, які виконує людина чи система.

3. Гейти (Gateways) – точки прийняття рішення або розгалуження процесів (наприклад, «так/ні»).

4. Потоки (Flows) – стрілки, що з'єднують елементи й показують послідовність.

5. Пули та доріжки (Pools & Lanes) – показують розподіл ролей/відповідальностей у процесі.

Приклад застосування BPMN в IT-проєктах:

Процес обробки заявки користувача:

- ✓ Користувач надсилає заявку.
- ✓ Система реєструє її в CRM.
- ✓ Менеджер перевіряє заявку.
- ✓ Якщо заявка валідна – передається в команду розробників.
- ✓ Якщо ні – заявка відхиляється.

Цей процес можна легко зобразити у BPMN-нотації як послідовність подій і завдань з умовним gateway («валідна/невалідна»).

Таблиця 8.8 – Спрощений BPMN-процес

Подія (Start Event)	Завдання (Task)	Рішення (Gateway)	Завершення (End Event)
Заявка надіслана	Реєстрація заявки в CRM	Валідна? Так/Ні	Прийнято або відхилено

8.3.2. UML (Unified Modeling Language) – уніфікована мова моделювання

UML – це стандартна мова моделювання, яка використовується для опису структури та поведінки програмних систем. Вона дозволяє розробникам, бізнес-аналітикам і менеджерам спільно бачити, як виглядає система, які в неї компоненти та як вони взаємодіють.

Типи UML-діаграм:

1. Структурні діаграми – показують, з чого складається система.

- ✓ Діаграма класів (Class Diagram): описує класи, їхні атрибути, методи

та зв’язки.

✓ Діаграма компонентів (Component Diagram): як частини системи з’єднані між собою.

2. Поводінкові діаграми – показують, як система поводиться у часі.

✓ Діаграма випадків використання (Use Case Diagram): взаємодія користувачів із системою.

✓ Діаграма послідовностей (Sequence Diagram): як об’єкти обмінюються повідомленнями під час виконання сценарію.

Приклад застосування UML в ІТ-проєктах (табл.8.9):

✓ Діаграма випадків (прецедентів) використання: показує, як користувач взаємодіє з інтернет-магазином (пошук товару, оформлення замовлення, оплата) (рис.8.1).

✓ Діаграма класів: описує сутності «Користувач», «Замовлення», «Товар» і їхні зв’язки (рис.8.2).

✓ Діаграма послідовностей: відображає процес авторизації користувача (введення логіну → перевірка в базі → підтвердження доступу) (рис.8.3-8.4).

Таблиця 8.9 – Опис спрощеної приклади UML-діаграм

Use Case Diagram	Class Diagram	Sequence Diagram
Користувач оформлює замовлення	Клас «Замовлення» пов’язаний із «Користувачем» та «Товаром»	Логін → Перевірка бази → Доступ
Користувач переглядає каталог	Клас «Каталог» містить список «Товарів»	Запит → Відповідь → Відображення даних

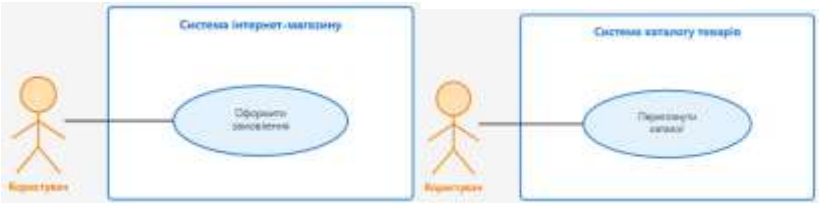

Рисунок 8.1 – Діаграма випадків

Рисунок 8.2 – Діаграма класів: Замовлення та Каталог

Рисунок 8.3 – Діаграма послідовностей (введення логіну)

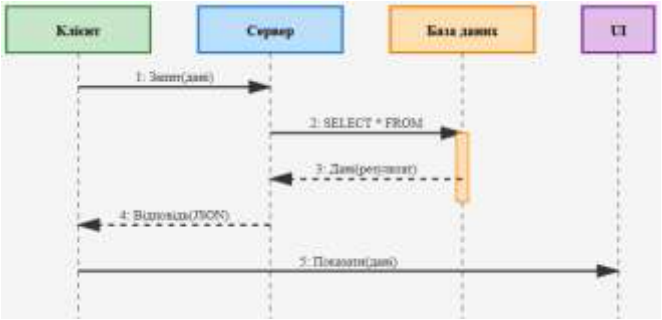

Рисунок 8.4 – Діаграма послідовностей (запит даних)

8.3.3. BPMN vs UML: коли який підхід краще застосовувати

1. BPMN (Business Process Model and Notation) – орієнтований на бізнес-процеси. Його характеристики:

✓ Зрозумілий не тільки розробникам, а й бізнес-стейкхолдерам.

✓ Використовується для опису потоків робіт, ролей, рішень.

✓ Ідеально підходить для бізнес-аналітиків, менеджерів, консалтингових компаній.

Приклад: процес обробки заявки в банку – від подання клієнтом до схвалення/відхилення.

2. UML (Unified Modeling Language) – орієнтований на архітектуру системи та технічні деталі. Його характеристики:

✓ Використовується розробниками для проектування ПЗ.

✓ Дозволяє описати класи, об’єкти, інтерфейси, а також сценарії роботи системи.

✓ Ідеально підходить для технічних спеціалістів: архітекторів, девелоперів, QA.

Приклад: створення діаграми класів для e-commerce-системи, де є сутності «Користувач», «Товар», «Замовлення».

Таблиця 8.10 – Порівняння BPMN і UML

Характеристика	BPMN	UML
Основна мета	Моделювання бізнес-процесів	Моделювання структури й поведінки систем
Орієнтація	Бізнес-користувачі, аналітики	Розробники, архітектори
Зрозумілість	Легка для бізнесу	Легка для технічних фахівців
Приклад використання	Процес обробки заявки	Діаграма класів для модуля замовлень
Тип задач	Опис потоку робіт, ролей	Опис компонентів, зв’язків, сценаріїв

8.4. Оцінка ефективності проєкту

Підрозділ містить узагальнено інструменти оцінки ефективності IT-проєктів.

8.4.1. KPI (Key Performance Indicators): швидкість команди, дефекти, своєчасність релізів

KPI (ключові показники ефективності) – це кількісні метрики, які дозволяють оцінити, наскільки успішно команда чи проєкт рухаються до поставлених цілей. У сфері IT-проєктів KPI допомагають менеджеру контролювати не тільки швидкість виконання робіт, але й якість кінцевого продукту.

Приклади KPI для IT-команд:

1. Швидкість команди (Velocity).

✓ Вимірюється кількістю story points, які команда виконує за один спринт.

✓ Дозволяє прогнозувати обсяг роботи на наступні ітерації.

2. Кількість дефектів (Defect Rate).

✓ Скільки багів знайдено під час тестування чи після релізу.

✓ Важливий показник якості продукту.

3. Своєчасність релізів (On-Time Delivery).

✓ Чи були ключові релізи доставлені у визначені строки.

✓ Дозволяє оцінити надійність планування та виконання.

4. Продуктивність (Throughput).

✓ Кількість виконаних завдань/issue за певний період.

✓ Добре видно на Kanban-дошках.

5. Задоволеність клієнта (Customer Satisfaction).

✓ Вимірюється через опитування чи NPS (Net Promoter Score).

✓ Дає уявлення про цінність продукту для користувачів.

Приклад із практики:

У продуктовій компанії KPI визначено так:

✓ Velocity = 45 story points/спринт.

✓ Defect Rate < 3 баги/1000 рядків коду.

✓ On-Time Delivery ≥ 90 %.

-Це дозволило керівництву бачити не тільки, скільки функціоналу команда встигає створити, а й наскільки якісно вона працює.

Таблиця 8.11 – KPI в ІТ-проєктах

KPI	Як вимірюється	Для чого використовується
Velocity	Story points за спринт	Прогнозування швидкості роботи команди
Defect Rate	Кількість багів/1000 рядків коду	Оцінка якості продукту
On-Time Delivery	Відсоток вчасних релізів	Контроль дотримання плану
Throughput	Кількість виконаних завдань за період	Вимірювання продуктивності
Customer Satisfaction	Опитування, NPS	Оцінка задоволеності користувачів

8.4.2. OKR (Objectives and Key Results): постановка цілей та вимірювання результатів

OKR – це сучасна методика постановки цілей, яку використовують провідні компанії світу (Google, Intel, LinkedIn). Вона допомагає не тільки визначати що ми хочемо досягти (Objectives), але й як ми вимірюємо прогрес Key Results).

Структура OKR:

1. Objective (Ціль): якісно описує бажаний результат, має бути амбітною, але досяжною.

Приклад: «Покращити досвід користувачів мобільного додатку».

2. Key Results (Ключові результати): конкретні кількісні показники, що доводять досягнення цілі.

Приклад:

- ✓ Знизити кількість крашів із 5 % до 1 %.
- ✓ Підняти середню оцінку у App Store з 4.0 до 4.5.
- ✓ Збільшити кількість активних користувачів на 20 %.

Відмінності OKR від KPI:

✓ KPI – це вимірювання поточної ефективності («як ми працюємо зараз»).

✓ OKR – це метод постановки стратегічних амбітних цілей («куди ми хочемо рухатися»).

В ідеалі вони працюють разом: KPI показують стан, а OKR – напрям розвитку.

Приклад із практики (табл.8.12):

У стартапі команда ставить Objective:

✓ «Вийти на ринок США з нашим SaaS-продуктом».

✓ Key Result 1: Отримати 1000 нових підписок за квартал.

✓ Key Result 2: Підписати 3 партнерські угоди.

✓ Key Result 3: Забезпечити 95 % uptime для американських користувачів.

Таблиця 8.12 – Objective → Key Results

Objective (Ціль)	Key Results (Ключові результати)
Вийти на ринок США з SaaS-продуктом	1. Отримати 1000 нових підписок за квартал 2. Підписати 3 партнерські угоди 3. Забезпечити 95 % uptime для американських користувачів
Покращити досвід користувачів мобільного додатку	1. Знизити кількість крашів із 5 % до 1 % 2. Підняти оцінку у App Store з 4.0 до 4.5 3. Збільшити кількість активних користувачів на 20 %

8.4.3. Відмінності KPI та OKR, їхнє комбіноване використання

1. KPI (Key Performance Indicators)

✓ Орієнтовані на стабільність і контроль.

✓ Вимірюють ефективність виконання поточних процесів.

✓ Допомагають зрозуміти, наскільки команда виконує план.

Приклад: Velocity = 45 story points/спринт, Defect Rate < 3/1000 рядків коду.

2. OKR (Objectives and Key Results)

✓ Орієнтовані на зростання та розвиток.

✓ Встановлюють амбітні цілі, що виходять за межі звичайних KPI.

✓ Дають напрям руху на стратегічному рівні.

Приклад: «Вийти на ринок США» + 3 ключові результати.

Таблиця 8.13 – Головні відмінності KPI та OKR

Характеристика	KPI	OKR
Основна мета	Контроль стабільності процесів	Постановка амбітних стратегічних цілей
Орієнтація	«Як ми працюємо зараз»	«Куди ми хочемо рухатися»
Масштаб	Тактичний, операційний рівень	Стратегічний рівень
Приклади	Velocity, Defect Rate, On-Time Delivery	Новий ринок, покращення UX, масштабування

Комбіноване використання KPI та OKR. Найефективніше їх поєднувати:

✓ KPI показують, наскільки добре команда справляється з поточними завданнями.

✓ OKR мотивують виходити за межі «звичного» і досягати нових результатів.

Приклад комбінування:

✓ KPI: On-Time Delivery ≥ 90 %.

✓ OKR: «Запустити нову мобільну платформу до кінця кварталу» + KR (1000 нових користувачів, 4.5 рейтинг в App Store).

Так компанія бачить і стабільність, і стратегічний розвиток.

8.4.4. Приклади метрик для стартапу та корпоративного ІТ-проєкту

1. Стартапи

У стартапах головний акцент робиться на зростанні та перевірці гіпотез. Метрики орієнтовані на ринок і користувачів.

✓ Customer Acquisition Cost (CAC) – скільки коштує залучення одного клієнта.

✓ Customer Lifetime Value (CLV) – яку загальну вигоду приносить клієнт за весь час співпраці.

✓ Monthly Active Users (MAU) – кількість активних користувачів на місяць.

✓ Churn Rate – відсоток користувачів, які перестають користуватися продуктом.

✓ Burn Rate – швидкість витрати інвестицій, «наскільки вистачить грошей».

Приклад (табл.8.14).

Стартап із мобільним застосунком відстежує:

✓ CAC = $10,

✓ CLV = $100,

✓ MAU = 20 000,

✓ Churn Rate = 8 % на місяць.

Таблиця 8.14 – Продукт має потенціал для масштабування.

Метрика	Що вимірює	Приклад
Customer Acquisition Cost (CAC)	Вартість залучення одного клієнта	$10 за клієнта
Customer Lifetime Value (CLV)	Загальна вигода від одного клієнта	$100 за весь час
Monthly Active Users (MAU)	Активні користувачі на місяць	20 000 MAU
Churn Rate	Відсоток відтоку користувачів	8 % на місяць
Burn Rate	Швидкість витрат інвестицій	На 12 місяців вистачає бюджету

2. Корпоративні IT-проєкти

У великих компаніях ключові метрики більше пов’язані з ефективністю процесів, якістю та стабільністю роботи системи.

✓ On-Time Delivery – % проєктів чи релізів, виконаних у строки.

✓ Defect Density – кількість дефектів на 1000 рядків коду.

✓ Mean Time to Repair (MTTR) – середній час на виправлення

критичного інциденту.

- ✓ System Uptime – час безвідмовної роботи (наприклад, 99.9 %).
- ✓ Employee Productivity – продуктивність команди (завдання/спринт).

Приклад (табл.8.15).

Корпоративний проєкт у банку відстежує:

- ✓ On-Time Delivery = 95 %,
- ✓ Defect Density = 1.5,
- ✓ MTTR = 4 години,
- ✓ Uptime = 99.95 %.

Таблиця 8.15 – Це свідчить про стабільність і високий рівень надійності.

Метрика	Що вимірює	Приклад
On-Time Delivery	Вчасність виконання релізів/проєктів	95 %
Defect Density	Кількість дефектів на 1000 рядків коду	1.5
Mean Time to Repair (MTTR)	Середній час усунення інциденту	4 години
System Uptime	Безвідмовна робота системи	99.95 %
Employee Productivity	Продуктивність команди	50 завдань/спринт

8.5. Загальні висновки

8.5.1. Основні тези лекції

✓ Сучасні інструменти (Jira, Trello, MS Project, Confluence) є основою організації командної роботи та прозорості процесів.

✓ Діаграми (BPMN, UML) дозволяють формалізувати бізнес-процеси та візуалізувати архітектуру систем.

✓ KPI та OKR допомагають оцінити ефективність роботи команди та рух до стратегічних цілей.

✓ Стартапи та корпорації застосовують різні метрики: для стартапів важлива швидкість росту й залучення користувачів, для корпорацій –

стабільність і якість.

✓ Інтеграція управління проєктами та бізнес-аналізу забезпечує баланс між технічними рішеннями й бізнес-цілями.

✓ Ключ до успіху – правильний вибір інструментів під конкретний проєкт і узгоджена робота між PM, BA та командою.

8.5.2. Питання для самоперевірки

1. Які переваги використання Jira порівняно з Trello?
2. Для чого використовується MS Project у великих організаціях?
3. У чому різниця між BPMN і UML?
4. Які показники можуть виступати KPI у команді розробки?
5. Чим KPI відрізняється від OKR?
6. Які метрики ефективності будуть більш важливими для стартапу, а які – для корпоративного проєкту?
7. Як Confluence допомагає у створенні та підтримці документації?
8. Чому інтеграція бізнес-аналізу й управління проєктами є ключем до успіху IT-проєктів?

8.6. Контрольні запитання

1. Чому сучасні інструменти управління проєктами є необхідністю, а не розкішшю? Обґрунтуйте на конкретних прикладах.
2. Що таке Jira? Опишіть основні функції цього інструменту для управління IT-проєктами.
3. Що таке Trello і для яких типів команд та проєктів він найбільш підходить?
4. У чому відмінність між Jira та Trello? Коли слід обирати кожен із цих інструментів?
5. Що таке Confluence і як він використовується для документування знань і вимог у проєкті?
6. Як системи контролю версій (Git, GitHub, GitLab) підтримують управління проєктами та змінами?
7. Що таке burn-down chart? Як він використовується для відстеження прогресу спринту?

8. Що таке velocity у контексті проєктного інструментарію та як цей показник розраховується і аналізується?

9. Що таке CI/CD (Continuous Integration / Continuous Delivery) і як ці практики пов'язані з управлінням проєктами?

10. Як автоматизація інтеграцій (наприклад, GitHub + Jira) знижує ручну роботу менеджера проєкту?

11. Що таке діаграма Ганта і в яких ситуаціях вона залишається корисним інструментом планування?

12. Опишіть три ключові тенденції в сучасних інструментах управління ІТ-проєктами: автоматизація, візуалізація, аналітика.

13. Які метрики продуктивності команди можна відстежувати за допомогою сучасних PM-інструментів?

14. Як розподілені команди використовують хмарні інструменти для спільної роботи в реальному часі?

15. Яким чином правильний вибір інструментів управління проєктом впливає на прозорість, співпрацю та кінцевий результат?

ПІДСУМКОК: ІНТЕГРАЦІЯ УПРАВЛІННЯ Й БІЗНЕС-АНАЛІЗУ ІТ-ПРОЄКТІВ

У посібнику розглянуто:

✓ Класичні методи управління (Waterfall, PMBOK) – акцент на формалізацію, контроль і передбачуваність.

✓ Гнучкі методи (Agile, Scrum, Kanban) – орієнтація на швидкі зміни, ітеративність і роботу з цінністю для користувача.

✓ Управління командою (ролі, динаміка за моделлю Такмана, конфлікти й комунікація зі стейкхолдерами).

✓ Управління ресурсами та ризиками (Gantt, CPM, матриця ризиків, SWOT, Монте-Карло).

✓ Управління змінами (ADKAR, модель Коттера, Change Requests).

✓ Бізнес-аналіз (збір вимог, трасування, управління змінами, використання RTM).

✓ Сучасні інструменти (Jira, Trello, MS Project, Confluence, Git).

Це формує системне бачення управління ІТ-проєктами від планування до реалізації.

Роль бізнес-аналітика у зв’язці з PM

✓ Project Manager відповідає за терміни, бюджет, ресурси та комунікації.

✓ Business Analyst забезпечує правильне формулювання вимог, зв’язок між замовником і технічною командою. Разом вони утворюють ключовий тандем, де PM тримає проєкт у рамках, а BA – гарантує, що кінцевий продукт відповідає бізнес-цілям.

Як інструменти допомагають реалізувати методології:

✓ Jira, Trello – підтримка Agile (спринти, Kanban-дошки).

✓ MS Project – підтримка Waterfall (Gantt, критичний шлях).

✓ Confluence – централізована база знань.

✓ Git/GitHub/GitLab – контроль версій, відстеження змін.

Інструменти не є цілями самі по собі, але вони дозволяють практично

втілити обрані методології.

Таблиця – Схема взаємодії: PM і BA

PM (Project Manager)	BA (Business Analyst)	Результат
Керує строками, бюджетом, ресурсами	Формалізує та відстежує вимоги	Продукт у строк і в бюджеті
Організовує комунікації	Забезпечує розуміння потреб замовника	Виконання бізнес-цілей
Координує команду	Підтримує життєвий цикл вимог	Якісний і цінний продукт

Висновки: інтеграція управління й бізнес-аналізу як ключ до успішних ІТ-проєктів:

✓ Управління без якісного бізнес-аналізу ризикує створити продукт «не той, що потрібен».

✓ Бізнес-аналіз без управління ризикує ніколи не завершитися через відсутність контролю.

✓ Тільки інтеграція підходів дозволяє створювати продукти, які одночасно:

- ✓ відповідають потребам користувачів,
- ✓ виконані вчасно й у бюджеті,
- ✓ якісні та масштабовані.

Таким чином, поєднання управлінських навичок і бізнес-аналітики – це стратегічна основа успіху сучасних ІТ-проєктів.

ПЕРЕЛІК СКОРОЧЕНЬ

API	Application Programming Interface – інтерфейс програмування застосунків
BA	Business Analyst – бізнес-аналітик
BABOK	Business Analysis Body of Knowledge – звід знань з бізнес-аналізу
BPMN	Business Process Model and Notation – нотація моделювання бізнес-процесів
BRD	Business Requirements Document – документ бізнес-вимог
CAC	Customer Acquisition Cost – вартість залучення клієнта
CAPM	Certified Associate in Project Management – сертифікований асоціат в управлінні проєктами
CBAP	Certified Business Analysis Professional – сертифікований професіонал з бізнес-аналізу
CCB	Change Control Board – рада контролю змін
CI/CD	Continuous Integration/Continuous Deployment – безперервна інтеграція/безперервне розгортання
CLV	Customer Lifetime Value – вартість клієнта протягом життєвого циклу
CMMI	Capability Maturity Model Integration – модель зрілості можливостей
CPI	Cost Performance Index – індекс виконання вартості
CPM	Critical Path Method – метод критичного шляху
CR	Change Request – запит на зміну
CRM	Customer Relationship Management – управління взаємовідносинами з клієнтами
CRUD	Create, Read, Update, Delete – створити, прочитати, оновити, видалити
DAD	Disciplined Agile Delivery – дисциплінована гнучка доставка
DevOps	Development and Operations – розробка та операції
DFD	Data Flow Diagram – діаграма потоків даних
DOORS	Dynamic Object Oriented Requirements System – динамічна

	об'єктно-орієнтована система вимог
ER	Entity-Relationship – сутність-зв'язок
ERP	Enterprise Resource Planning – планування ресурсів підприємства
EV	Earned Value – освоєний обсяг
EVM	Earned Value Management – управління освоєним обсягом
FR	Functional Requirements – функціональні вимоги
GDPR	General Data Protection Regulation – загальний регламент захисту даних
HTML	HyperText Markup Language – мова гіпертекстової розмітки
IIBA	International Institute of Business Analysis – міжнародний інститут бізнес-аналізу
ISO	International Organization for Standardization – міжнародна організація зі стандартизації
ITIL	Information Technology Infrastructure Library – бібліотека інфраструктури інформаційних технологій
KPI	Key Performance Indicators – ключові показники ефективності
MAU	Monthly Active Users – щомісячні активні користувачі
ML	Machine Learning – машинне навчання
MS Project	Microsoft Project – програмний продукт для управління проєктами
MTTR	Mean Time to Repair – середній час відновлення
MVP	Minimum Viable Product – мінімально життєздатний продукт
NFR	Non-Functional Requirements – нефункціональні вимоги
NPS	Net Promoter Score – індекс лояльності споживачів
NPV	Net Present Value – чиста приведена вартість
OGC	Office of Government Commerce – офіс урядової комерції
OKR	Objectives and Key Results – цілі та ключові результати
PAN	Primary Account Number – основний номер рахунку
PCI DSS	Payment Card Industry Data Security Standard – стандарт безпеки даних індустрії платіжних карт
PDCA	Plan-Do-Check-Act – планувати-виконувати-перевіряти-діяти
PERT	Program Evaluation and Review Technique – техніка оцінювання

	та перегляду програм
PESTEL	Political, Economic, Social, Technological, Environmental, Legal – політичні, економічні, соціальні, технологічні, екологічні, правові (фактори)
PM	Project Manager – проєктний менеджер
PMBOK	Project Management Body of Knowledge – звід знань з управління проєктами
PMI	Project Management Institute – інститут управління проєктами
PMO	Project Management Office – офіс управління проєктами
PMP	Project Management Professional – професіонал з управління проєктами
PO	Product Owner – власник продукту
PoC	Proof of Concept – підтвердження концепції
PRINCE2	Projects IN Controlled Environments – проєкти в контрольованому середовищі
PV	Present Value – приведена вартість
QA	Quality Assurance – забезпечення якості
RACI	Responsible, Accountable, Consulted, Informed – відповідальний, підзвітний, консультований, інформований
RFP	Request for Proposal – запит на пропозицію
RFQ	Request for Quotation – запит на цінову пропозицію
ROI	Return on Investment – повернення інвестицій
RTM	Requirements Traceability Matrix – матриця простежуваності вимог
SAFe	Scaled Agile Framework – масштабований гнучкий фреймворк
SDLC	Software Development Life Cycle – життєвий цикл розробки програмного забезпечення
SLA	Service Level Agreement – угода про рівень обслуговування
SM	Scrum Master – скрам-майстер
SMART	Specific, Measurable, Achievable, Relevant, Time-bound – конкретний, вимірюваний, досяжний, релевантний, обмежений у часі
SPI	Schedule Performance Index – індекс виконання графіка

SQL	Structured Query Language – мова структурованих запитів
SRS	Software Requirements Specification – специфікація вимог до програмного забезпечення
SWOT	Strengths, Weaknesses, Opportunities, Threats – сильні сторони, слабкі сторони, можливості, загрози
TKI	Thomas-Kilmann Instrument – інструмент Томаса-Кілманна
TRL	Technology Readiness Level – рівень готовності технології
UAT	User Acceptance Testing – приймальне тестування користувачами
UI	User Interface – користувацький інтерфейс
UML	Unified Modeling Language – уніфікована мова моделювання
UX	User Experience – користувацький досвід
WBS	Work Breakdown Structure – структура декомпозиції робіт
WCAG	Web Content Accessibility Guidelines – настанови з доступності вебконтенту
WIP	Work In Progress – робота в процесі
WSJF	Weighted Shortest Job First – зважений найкоротший процес спочатку

СПИСОК ЛІТЕРАТУРИ

1. Schwalbe K. Information Technology Project Management. 10th Edition. Boston : Cengage, 2023. 640 p.

2. Verzuh E. The Fast Forward MBA in Project Management. 7th Edition. Hoboken, NJ : Wiley, 2024. 512 p.

3. Heldman K. PMP Project Management Professional Exam Study Guide: 2024-2025 Edition. Hoboken, NJ : Wiley, 2024. 800 p.

4. Harvard Business Review Project Management Handbook. Boston : Harvard Business Review Press, 2024. 304 p.

5. Olabode S. A. O. The Agile Project Manager: A Complete Guide to Mastering Agile Project Management. Independently published, 2024. 130 p.

6. Turner R. J. Gower Handbook of Project Management. 7th Edition. London : Routledge, 2024. 720 p.

7. Cooper D. Project Management: A Practical Guide. London : Routledge, 2024. 320 p.

8. Lledó P. The Project Manager: A Guide to Mastering the Art of Project Management. New York : Apress, 2024. 280 p.

9. H. B. Modern IT Project Management: A Practical Guide to Agile, Scrum, and DevOps. Independently published, 2023. 156 p.

10. Agrawal P. PMP Exam Prep 2024-2025: A Complete Guide to PMP Certification. Independently published, 2024. 250 p.

11. Egeland M. PMP Exam Prep 2024: A Complete Guide. Independently Published, 2024. 312 p.

12. Hamilton A. IT Project Management for Beginners. Independently published, 2023. 119 p.

13. Cohn M. Agile Estimating and Planning (Addison-Wesley Signature Series). Boston : Addison-Wesley, 2023. 368 p.

14. Schmidt T. Strategic Project Management: Leading Change and Transformation. London : Routledge, 2024. 312 p.

15. Wysocki R. K. Effective Project Management: Traditional, Agile, Extreme, Hybrid. 9th Edition. Hoboken, NJ : Wiley, 2024. 704 p.

16. Lewis J. DevOps for Project Managers: A Practical Guide. Independently

published, 2024. 160 p.

17. Wallace M. C. Scrum: The Ultimate Guide to Mastering Scrum in IT Project Management. Independently published, 2023. 104 p.

18. Parker S. The Agile Mindset: A Practical Guide to Agile Project Management. Independently published, 2023. 132 p.

19. Williams D. E. Project Management for Humans: A Practical Guide to Getting Things Done. Independently published, 2023. 150 p.

20. Cadle J., Paul D., Turner P. Business Analysis Techniques: 123 Essential Tools for Success. 4th Edition. London : BCS, The Chartered Institute for IT, 2023. 608 p.

21. Larson E., Larson R. CBAP / CCBA Certification Study Guide. 2nd Edition. Hoboken, NJ : Wiley, 2024. 512 p.

22. Pohl K. Requirements Engineering: A Practitioner's Guide. Berlin : Springer, 2024. 450 p.

23. Hughes S. The Agile Business Analyst: A Guide to Business Analysis in Agile Projects. Independently published, 2023. 142 p.

24. S. J. D. Business Analysis for Beginners: A Practical Guide to Business Analysis in IT. Independently published, 2023. 125 p.

25. Porter M. Business Analysis in the Digital Age: A Guide to BA in IT and Digital Transformation. Independently published, 2023. 170 p.

26. Scott R. Strategic Business Analysis for IT: Aligning IT with Business Goals. Independently published, 2024. 180 p.

27. Sharma P. Software Requirements: A Practical Guide to Elicitation and Validation. New Delhi : BPB Publications, 2024. 250 p.

28. Brooks R. Mastering User Stories: A Guide to Requirements in Agile. Independently published, 2023. 130 p.

29. Blair R. The Business Analyst's Toolkit: A Practical Guide to Essential BA Techniques. Independently published, 2025. 190 p.

30. K. A. Requirements Engineering for Agile Projects: A Practical Guide. Independently published, 2023. 155 p.

31. Williams S. UML for Business Analysts: A Practical Guide. Independently published, 2023. 130 p.

32. James T. Modern Business Analysis: A Practical Guide for Digital

Transformation. Independently published, 2023. 165 p.

33. McGreal D., Jocham R. The Professional Product Owner: Leveraging Scrum as a Competitive Advantage. 3rd Edition. Boston : Addison-Wesley, 2023. 336 p.

34. Olsen D. The Lean Product Playbook: How to Innovate with Minimum Viable Products and Rapid Customer Feedback. 2nd Edition. Hoboken, NJ : Wiley, 2023. 320 p.

35. LeMay M. Product Management in Practice: A Real-World Guide to the Key Connective Role. 2nd Edition. Sebastopol, CA : O'Reilly, 2023. 288 p.

36. Lombardo C. T. Product Roadmaps Relaunched: How to Set Direction while Embracing Uncertainty. Sebastopol, CA : O'Reilly, 2023. 250 p.

37. Olson T. The Product Led Organization: Drive Growth by Putting Product at the Center. Hoboken, NJ : Wiley, 2023. 256 p.

38. Miller L. Product Ownership in Practice: A Guide to Mastering the Product Owner Role. Independently published, 2024. 175 p.

39. Thomas S. The Product Owner's Survival Guide: A Practical Guide to Product Ownership. Independently published, 2025. 160 p.

40. Cagan M. Inspired: How to Create Tech Products Customers Love. 2nd Edition (New Printing). Hoboken, NJ : Wiley, 2023. 368 p.

41. Erickson T. The Product Manager's Guide to Growth: A Practical Handbook. Independently published, 2024. 210 p.

42. Evans M. Product Strategy for the Digital Age: A Practical Guide. . Independently published, 2024. 180 p.

43. Costa D. A. AI in Project Management: How to Use AI for IT Project Success. Independently published, 2024. 150 p.

44. Smith J. AI-Powered Project Management: A Guide for 2025. FutureTech Press, 2024. 220 p.

45. Lee H. Generative AI for Project Managers: A Practical Guide. Independently published, 2024. 140 p.

46. Edwards M. The AI Project Manager: Leading AI and Data Science Projects. Apress, 2023. 315 p.

47. H. M. IT Governance: A Practical Guide to Implementing COBIT and ITIL. Independently published, 2023. 180 p.

48. Green T. Digital Transformation Strategy: A Practical Guide for IT Leaders. Independently published, 2024. 200 p.

49. Ткаченко Р. О. Управління ІТ-проектами : навчальний посібник. Київ : КПІ ім. Ігоря Сікорського, 2023. 250 с.

50. Інститут управління проєктами. Настанова до Зводу знань з управління проєктами (Настанова PMBOK). 7-е вид. / пер. з англ. Київ : PMI Ukraine Chapter, 2023. 360 с.

51. Сергєєв О. А. Бізнес-аналіз : практикум. Київ : ЛІРА-К, 2023. 320 с.

52. Сазерленд Д. Scrum. Навчись робити вдвічі більше за менший час / пер. з англ. О. Асташової. Харків : КСД, 2024. 288 с.

53. Коллінз Д. Від хорошого до величного / пер. з англ. Р. Скакун. Київ : Наш Формат, 2023. 344 с.

ДОДАТОК А.
ЗВЕДЕНІ ТАБЛИЦИ

Таблиця А.1 – Порівняльна таблиця провалів через погане планування в IT-проєктах

Причина	Приклад	Наслідок	Висновок
Недостатній бюджет	Стартап із маркетплейсом: грошей вистачило тільки на MVP.	Продукт виглядав незавершеним, інвестори відмовили.	Потрібно чітко планувати бюджет і пріоритети.
Вигорання команди	PM перевантажив Dev команду понаднормовою роботою.	Ключові розробники звільнилися, швидкість розробки впала.	Перевантаження у короткій перспективі шкодить довгостроково.
Технічні збої	Банківський проєкт без резервних серверів.	Система «лягла», клієнти кілька днів без доступу, втрати грошей і репутації.	Треба планувати масштабування й резерви.
Denver Airport Baggage System (1990-ті)	Автоматизація багажу.	Втрати понад $500 млн через зрив запуску.	Недооцінка складності й відсутність аналізу ризиків.
NHS IT Programme (2000-ті)	Національна система охорони здоров’я Великої Британії.	Закрито після витрат £10 млрд.	Помилки управління ресурсами, відсутність контролю.

Таблиці А.2 – Методи оцінки потреб у ресурсах (Метод – Суть – Приклад – Переваги – Недоліки)

Метод	Суть	Приклад	Переваги	Недоліки
Expert Judgment (експертна оцінка)	Залучення досвідчених спеціалістів для прогнозу ресурсів.	Техлід оцінює, що потрібні 2 Dev на 3 місяці.	Швидко, враховує досвід.	Суб’єктивність.
Аналогове оцінювання	Порівняння з попередніми схожими проєктами.	Новий мобільний застосунок оцінюється як попередній (6 міс., 5 людей).	Простота, використання історичних даних.	Точність залежить від схожості проєктів.
Параметричне оцінювання	Математичні моделі на основі статистики.	50 функцій × 20 годин = 1000 годин.	Об’єктивність, кількісний підхід.	Потрібні дані й статистика.
Bottom-Up Estimation (знизу-вгору)	Оцінка кожного дрібного завдання з подальшою сумою.	Окрема оцінка фронтенду, бекенду, дизайну, QA.	Найточніший метод.	Витратний за часом.
Planning Poker (Agile)	Командне оцінювання завдань у story points.	Команда дає оцінки складності функцій.	Враховує думку всієї команди, підвищує залученість.	Суб’єктивність, не завжди точний прогноз.

Таблиця А.3 – Порівняльна таблиця інструментів для планування та розподілу ресурсів

Інструмент	Основні можливості	Сильні сторони	Виклики
MS Project	Діаграми Ґанта, критичний шлях, управління бюджетом та ресурсами.	Глибока деталізація, інтеграція з Microsoft 365, підходить для великих проєктів.	Складний у навчанні, потрібна ліцензія.
Jira	Backlog, спринти, scrum- і kanban-дошки, аналітика.	Гнучкість, інтеграції (Confluence, Bitbucket), автоматизація процесів.	Перевантажена для маленьких команд.
Trello	Kanban-дошки, списки й картки для управління завданнями.	Інтуїтивний інтерфейс, швидке впровадження, безкоштовна версія.	Обмежені можливості для масштабних проєктів.
Resource Guru	Планування завантаження співробітників, календар доступності, управління відпустками.	Проста візуалізація workload, інтеграція з Google Calendar, Outlook.	Немає повного функціоналу управління проєктами (тільки ресурси).

Таблиця А.4 – Зведена таблиця технічних ризиків

Ризик	Причина	Наслідок	Як мінімізувати
Баги у коді	Недостатнє тестування, відсутність code review.	Збої на продакшені, падіння системи.	Code Review, автоматизоване тестування, CI/CD.
Несумісність технологій	Невдалий вибір бібліотек або API.	Функціонал не працює, інтеграції ламаються.	Proof of Concept (PoC), попередній технічний аналіз.
Застарілі технології	Використання фреймворків без підтримки.	Проблеми з безпекою, неможливість оновлень.	Регулярне оновлення стеку, аналіз життєвого циклу технологій.
Проблеми з продуктивністю	Неправильна оцінка навантажень.	Система не витримує пікових користувачів.	Навантажувальне тестування, оптимізація архітектури.
Зміни у сторонніх сервісах	API сторонніх систем змінюється без попередження.	Ламаються інтеграції, зрив роботи продукту.	Моніторинг API, fallback-механізми, регулярні рев'ю інтеграцій.

Таблиця А.5 – Зведена таблиця бізнес-ризиків

Ризик	Причина	Наслідок	Як мінімізувати
Зміна вимог	Замовник змінює бачення продукту.	Доопрацювання, перевитрати бюджету, затримки.	Agile-ітерації, Change Management, фіксація змін у документації.
Втрати клієнта	Компанія втрачає ключового користувача або замовника.	Замороження чи зупинка проєкту.	Диверсифікація клієнтів, довгострокові контракти.
Зміна ринку	Поява конкурентів із кращим продуктом.	Зниження цінності продукту.	Регулярний аналіз ринку, оновлення стратегії продукту.
Зміна регуляцій	Нові закони (GDPR, податки, безпека).	Необхідність термінових змін у системі.	Юридичний моніторинг, закладання часу/бюджету на відповідність.
Фінансові проблеми замовника	Скорочення бюджету чи закриття бізнесу.	Зупинка фінансування проєкту.	Перевірка фінансової стабільності клієнта, поетапне фінансування.

Таблиця А.6 – Зведена таблиця організаційних ризиків

Ризик	Причина	Наслідок	Як мінімізувати
Висока плинність кадрів	Ключові спеціалісти залишають команду.	Втрати знань, затримки у роботі.	Документування знань, план наступності (knowledge transfer).
Конфлікти в команді	Неефективна комунікація, відсутність лідерства.	Зниження продуктивності, поганий клімат у команді.	Фасилітація, медіація, ретроспективи.
Неефективний менеджмент	Погане планування, відсутність контролю.	Затримки, перевитрати бюджету.	Навчання менеджерів, застосування методологій управління проєктами.
Низька кваліфікація команди	Недостатній досвід чи навички у розробників.	Баги, перевищення термінів.	Навчання, менторство, перевірка компетенцій на старті.
Проблеми з комунікаціями	Відсутність каналів або їхнє неправильне використання.	Неправильне розуміння вимог, помилки.	Використання сучасних інструментів (Slack, Jira, Confluence), регламент комунікацій.

Таблиця А.7 – Зведена таблиця зовнішніх ризиків

Ризик	Причина	Наслідок	Як мінімізувати
Зміни в законодавстві	Нові вимоги до безпеки даних, ліцензування.	Потреба в термінових змінах у системі, додаткові витрати.	Юридичний моніторинг, закладення резерву бюджету й часу.
Економічна нестабільність	Інфляція, коливання валют.	Збільшення бюджету, зниження прибутковості.	Хеджування ризиків, резервний бюджет.
Політичні фактори	Війна, санкції, обмеження експорту технологій.	Зупинка або перенесення проєкту, втрата ринків.	Альтернативні ринки, розподілені команди, план безперервності бізнесу.
Форс-мажори	Пандемії, катастрофи, відключення енергії.	Призупинення роботи, затримки у виконанні завдань.	Резервні канали живлення, віддалена робота, disaster recovery plan.
Ринкові ризики	Зміна трендів, падіння попиту.	Продукт втрачає актуальність.	Аналіз ринку, регулярне оновлення стратегії продукту.

Таблиця А.8 – Приклади кейсів ризиків

Тип ризику	Ситуація	Наслідок	Урок
Технічний	У банківській системі некоректний алгоритм округлення призвів до неправильного розрахунку відсотків.	Банк зазнав мільйонних збитків через компенсації клієнтам.	Потрібно проводити автоматизоване тестування та перевірку критичних сценаріїв.
Бізнесовий	Стартап для подорожей втратив інвестора під час пандемії COVID-19 через падіння попиту.	Проєкт заморозили на 2 роки, команда розпущена.	Необхідно мати альтернативний бізнес-план і гнучкість у зміні моделі.
Організаційний	У продуктовій компанії ключовий архітектор залишив проєкт посеред розробки.	Затримка на 6 місяців через відсутність документації та knowledge transfer.	Потрібно мати план передачі знань і підтримувати мотивацію співробітників.
Зовнішній	Українська ІТ-компанія стикалась із відключеннями електроенергії під час війни 2022-2023 рр.	Робота зупинялась на кілька годин або днів.	Використання генераторів, UPS та віддалених команд зменшує ризик простою.

РОЗШИРЕНИЙ ЗМІСТ

Навчальне видання

ГЛАВЧЕВ Максим Ігорович
ГЛАВЧЕВ Дмитро Максимович

УПРАВЛІННЯ ТА БІЗНЕС-АНАЛІЗ ІТ-ПРОЄКТІВ

Навчальний посібник
для студентів спеціальності
F7 «Комп'ютерна інженерія»
денної та заочної форм навчання

Відповідальний за випуск проф. Заковоротний О. Ю.

Роботу до видання рекомендував проф. Заполовський М. Й.

В авторській редакції

План 2026 р., поз. 14

Гарнітура Times New Roman. Ум. друк. арк. 10.0

Видавничий центр НТУ «ХПІ».
Свідоцтво про державну реєстрацію ДК № 5478 від 21.08.2017 р.
61002, м. Харків, вул. Кирпичова, 2.

Електронне видання