## Лекція 2

# Інженерія вимог

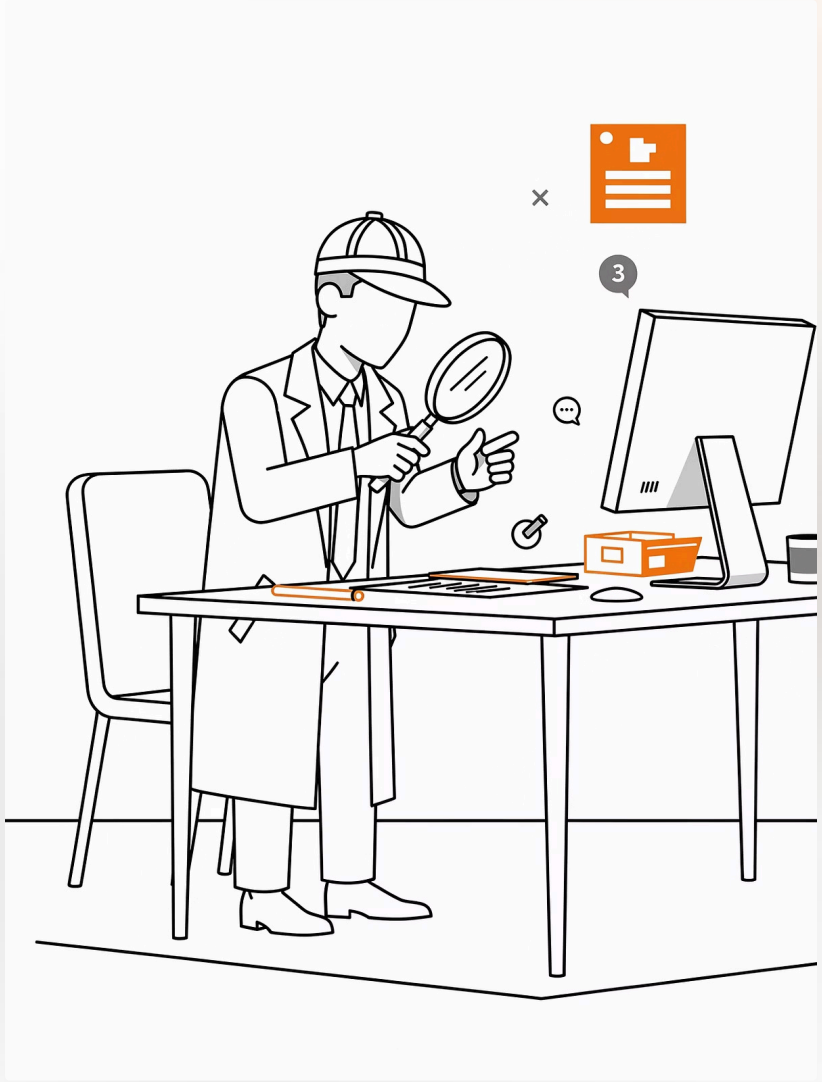
# Виявлення vs. Збір вимог

Більшість початківців каже "збір вимог", уявляючи, що вимоги лежать на поверхні, як гриби в лісі. Професійна реальність інша: вимоги приховані глибоко у свідомості замовника, часто замасковані під "хотілки", суперечливі або взагалі неусвідомлені.

Тому ми використовуємо термін **Виявлення (Elicitation)** — процес, схожий на роботу детектива або лікаря-діагноста.

> "Замовник не знає, чого він хоче. Замовник знає, що у нього болить."

Ваше завдання — знайти цю "біль" і запропонувати "ліки" (ПЗ), а не просто записати під диктовку "зробіть мені червону кнопку".



# Метод №1: Інтерв'ю

### Структуроване

Чіткий список питань. Нічого не забудете, але нагадує допит.

### Неструктуроване

Вільна бесіда. Комфортно для клієнта, але можна проговорити 2 години без результату.

### Напівструктуроване

Рекомендоване. Є ключові теми, але можна відхилятися для інсайтів.

### Техніка "5 Why"

Клієнт: "Хочу експорт в Excel"

Ви: Чому?

Клієнт: "Щоб передати бухгалтеру"

Ви: Чому вручну?

Клієнт: "Бо немає інтеграції з 1С"

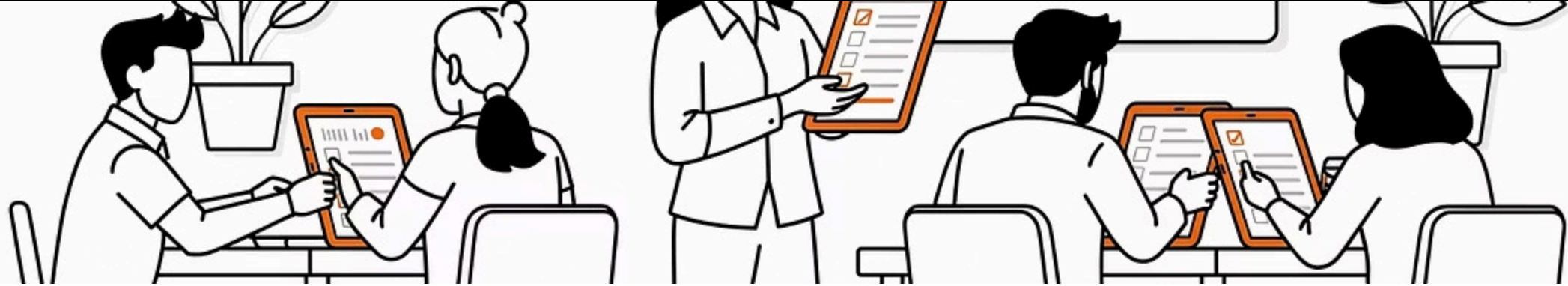
**Висновок:** Справжня вимога — інтеграція з 1С, а не кнопка Excel.

### The Mom Test

Не питайте думку про майбутнє — люди брешуть з ввічливості.

~~Погано:~~ "Вам подобається ідея додатку для фітнесу?"

Добре: "Коли ви востаннє займалися спортом? Як відстежували прогрес? Що вас бісило?"



# Метод №2: Анкетування

Якщо інтерв'ю — це "глибина" (якісні дані), то анкетування — це "ширина" (кількісні дані). Використовується, коли майбутніх користувачів тисячі.

### Коли застосовувати:

- Потрібно підтвердити гіпотезу на великій вибірці
- Користувачі географічно розкидані
- Потрібна статистика (наприклад, версії Android)

#### Selection Bias

Помилка вибірки. Опитування Senior Java розробників не розкаже про проблеми студентів-першокурсників.

#### Некоректні питання

Закриті питання — легко аналізувати, мало інформації. Відкриті — багато інформації, але ніхто не любить писати твори.

#### Навідні питання

"Чи згодні ви, що наш інтерфейс зручний?" — це маніпуляція, а не дослідження.

# Метод №3: Аналіз конкурентів

Найшвидший спосіб сформувати вимоги — подивитися, як цю проблему вирішили інші. В індустрії це називається **Benchmarking**.

### Навіщо це робити:

- Не винаходити велосипед
- Користувачі звикли до певних патернів
- Знайти "Killer Feature"

### Що аналізуємо:

**Прямі конкуренти:** Вирішують ту саму проблему тим самим способом (Uber vs Uklon)

**Непрямі конкуренти:** Вирішують ту саму проблему іншим способом (Uber vs Громадський транспорт)

01

### Пройти User Journey

Досліджуємо шлях користувача в додатку конкурента

02

### Виписати функції

Складаємо список функцій, які забезпечують цей шлях

03

### Проаналізувати відгуки

Негативні відгуки в AppStore/Google Play — скарбниця вимог. Користувачі самі пишуть, чого не вистачає

# Від документа до розмови

Класичний підхід (IEEE 830 SRS) пропонував писати: "Система повинна надавати можливість автентифікації...". Це сухо, нудно і фокусується на системі, а не на людині.

**Agile змінив парадигму.** Ми більше не пишемо вимоги для "системи". Ми описуємо історії живих людей.

> **User Story (Історія користувача)** — це короткий, простий опис функції з точки зору людини, яка бажає отримати нову можливість. Це не детальне ТЗ. Це запрошення до обговорення.

# Модель "3C": Анатомія історії

Рон Джеффріс (один із авторів Agile Manifesto) описав структуру User Story моделлю **3C**:

### Card (Картка)

Фізичний стікер або тікет у Jira. Містить заголовок і короткий опис. Нагадує, що є потреба.

### Conversation (Розмова)

Найважливіша частина. Команда обговорює деталі. Деталі народжуються в діалозі.

### Confirmation (Підтвердження)

Критерії приймання. Умови, за яких історія вважається виконаною.

# Шаблон User Story

### Стандартний формат

**Як** <Роль / Персона>,

**Я хочу** <Дія / Функціонал>,

**Щоб** <Цінність / Мотивація>.

### Приклад:

*Як студент університету,*

*Я хочу отримувати сповіщення про зміни в розкладі на телефон,*

*Щоб не приходити на пари, які скасували, і поспати довше.*

### Типові помилки:

~~"Як користувач, я хочу логінитися..."~~

Занадто абстрактна роль "користувач"

~~"...Я хочу, щоб кнопка була синя"~~

Це дизайн, а не історія

~~"...Щоб користуватися системою"~~

Це не цінність. Цінність має бути вимірюваною

# Критерії якості: INVEST

Як зрозуміти, що ваша історія хороша? Використовуйте мнемоніку **INVEST**, розроблену Біллом Вейком.

### Independent

Незалежна. Історію можна зробити окремо від інших.

### Negotiable

Обговорювана. Це не наказ, а тема для торгу.

### Valuable

Цінна. Приносить користь бізнесу або клієнту.

### Estimable

Оцінювана. Команда може сказати, скільки часу це займе.

### Small

Маленька. Має вміщуватися в один Спринт (2-3 дні роботи).

### Testable

Тестована. QA інженер може написати тест-кейс.

# Критерії приймання (Acceptance Criteria)

Якщо User Story — це "Хотілка", то AC — це "Умови контракту". Це єдине джерело правди про те, як саме має працювати фіча.

### Навіщо потрібні AC:

- Визначають межі (Scope)
- Допомагають QA писати тести до початку розробки
- Захищають від "Gold plating"

### Формат А: Простий список

**Історія:** Форма реєстрації

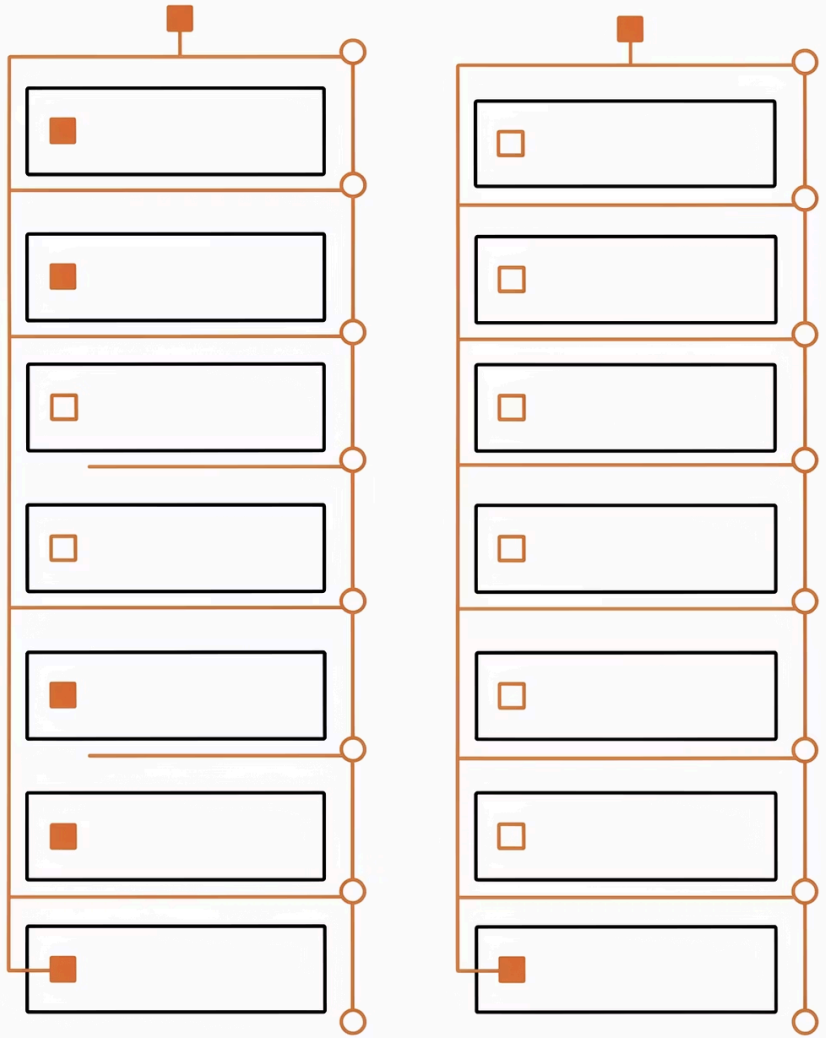
**AC:**

- Поле "Email" обов'язкове
- Пароль прихований зірочками
- Кнопка неактивна, доки поля пусті

### Формат Б: Gherkin (Given / When / Then)

```
Scenario: Успішний вхід в систему
Given: Користувач на сторінці логіну
And: Має акаунт "user@test.com" з паролем "1234"
When: Він вводить "user@test.com" та "1234"
And: Натискає кнопку "Увійти"
Then: Система перенаправляє на "Головну сторінку"
And: Показує повідомлення "Вітаємо, User!"
```

Цей формат дозволяє автоматично генерувати автотести (Cucumber/SpecFlow).



# DoD vs AC: Не плутати!

### Acceptance Criteria (AC)

**Унікальні для кожної історії**

Для логіну — перевірка пароля

Для кошика — підрахунок суми

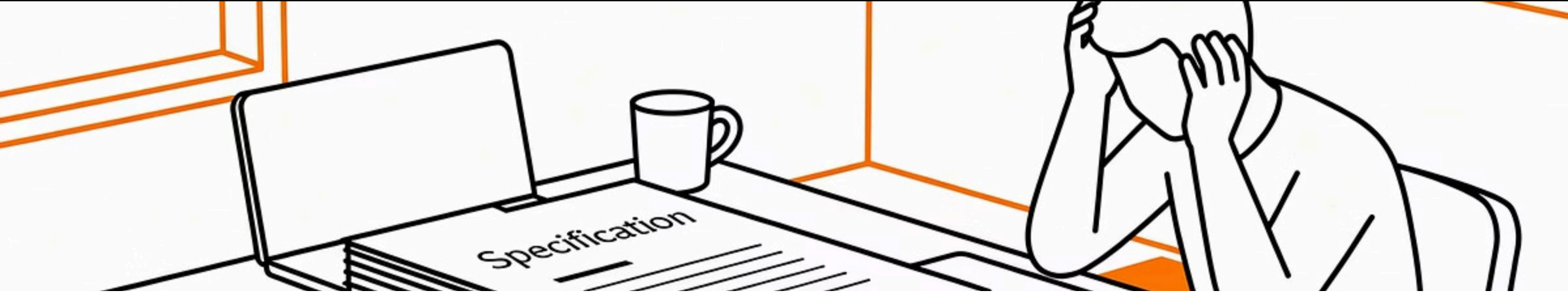
Для пошуку — фільтрація результатів

### Definition of Done (DoD)

**Єдиний чек-ліст для ВСІХ історій**

- Код написаний
- Код пройшов Review
- Автотести зелені
- Документація оновлена

Якщо AC виконані, але DoD — ні (наприклад, забули оновити документацію), історія не приймається.




# Ілюзія контролю

У класичному менеджменті панує переконання: "Якщо ми напишемо дуже детальне Технічне Завдання (ТЗ) на 100 сторінок, де опишемо кожну кнопку, то розробники зроблять саме те, що треба".

На практиці це працює лише для простих задач. Для складних систем це ілюзія.

> **Парадокс ТЗ:**
>
> Чим детальніше ви описуєте систему на старті, тим більше помилок ви закладаєте в фундамент, оскільки на старті ви знаєте про продукт найменше.

# Agile-підхід: User Stories + Backlog

В Agile поняття "ТЗ" розмите. Його замінює **Product Backlog** — живий список вимог.

### Динамічність

Беклог постійно змінюється. Вимоги деталізуються тільки перед тим, як їх брати в роботу (Just-in-Time).

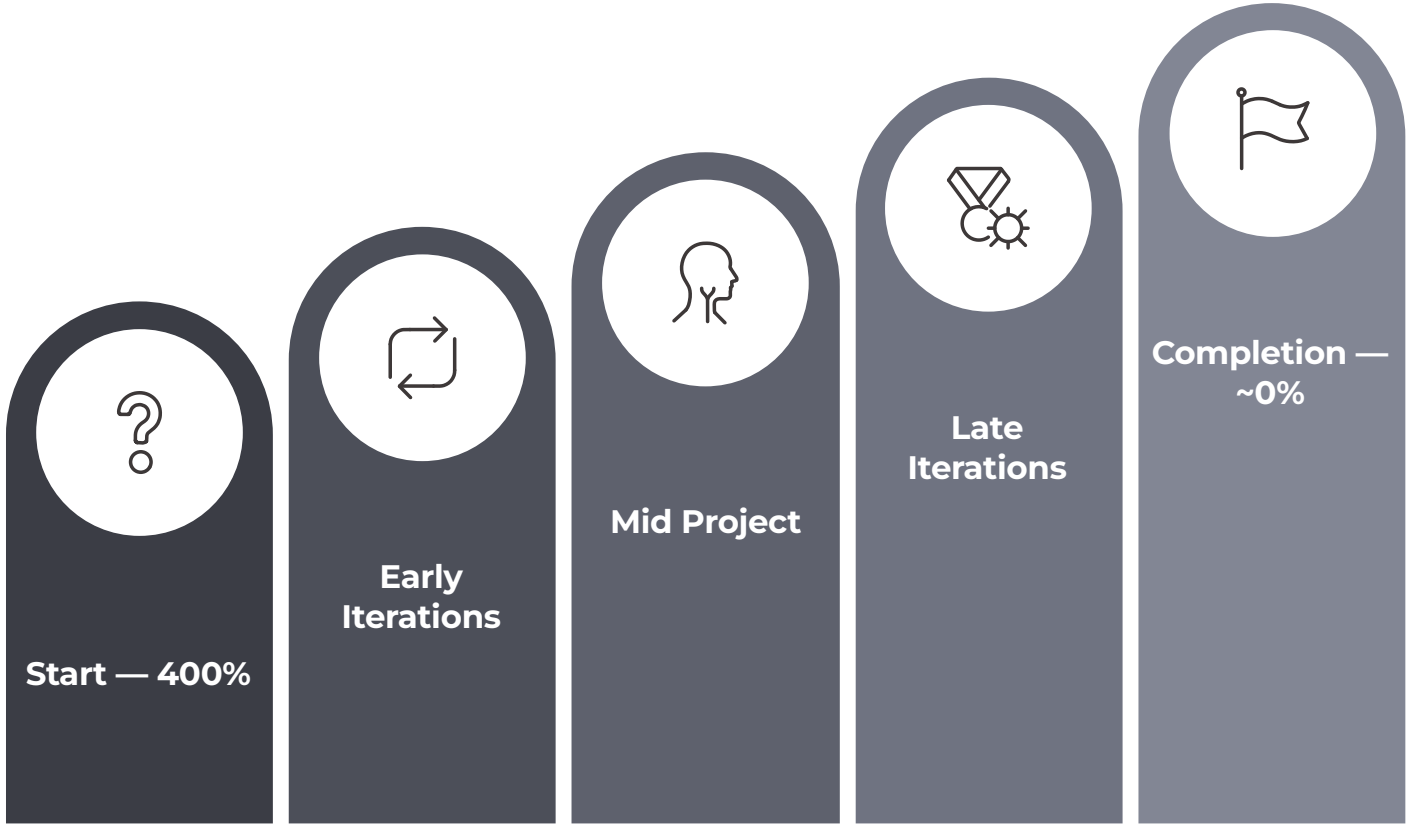
### Орієнтація на проблему

User Story каже, яку проблему треба вирішити, а рішення придумує команда.

### Фрагментарність

Немає одного файлу "ТЗ.docx". Є набір тікетів у Jira/Trello.

# Конус невизначеності




На початку проєкту (стадія ініціації) похибка в оцінці обсягу та часу може сягати **400% (4x)**.

Класичне ТЗ намагається зафіксувати вимоги в точці максимальної невизначеності. Це гра в рулетку.

Agile визнає цю невизначеність і звужує конус поступово, ітерація за ітерацією.

# Порівняльна таблиця: ТЗ vs User Stories

| Критерій | Класичне ТЗ | User Stories (Agile) |
|---|---|---|
| Розмір | Десятки/сотні сторінок | Картка + Коментарі + Тест-кейси |
| Коли пишеться | До початку кодингу (Up-front) | Протягом проєкту (Just-in-Time) |
| Хто пише | Аналітик (одноосібно) | Команда + Product Owner |
| Деталізація | Максимальна одразу | Спочатку низька, деталізується перед спринтом |
| Ціль | Юридичний контракт | Комунікація, спільне розуміння |
| Вартість змін | Висока (переписати, перепогодити) | Низька (викинути стікер, написати новий) |
| Ризик | Зробити "точно за ТЗ", але непотрібне | Зробити не те (якщо погана комунікація) |

# Сучасний компроміс: Discovery Phase

У реальному аутсорсингу чистий Agile (без документації) продати важко, а чистий Waterfall (ТЗ на рік) — ризиковано. Тому виник стандартний етап — **Discovery Phase**.

Це "нульовий етап" проєкту (2-4 тижні), де аналітики та архітектори:

**1** **Створюють High-Level Vision**

Загальне бачення продукту та його цілей

**2** **Малюють прототипи**

UI/UX дизайн ключових екранів

**3** **Пишуть початковий Беклог**

User Stories для MVP (Minimum Viable Product)

**4** **Фіксують архітектуру**

C4 діаграми Level 1-2

Це дозволяє отримати точність оцінки, достатню для старту, але не витрачати пів року на написання "мертвого" ТЗ.

# Ієрархія вимог у Беклозі

У сучасному проєкті вимоги не лежать купою. Вони структуровані ієрархічно.

### Epic (Епік)

Глобальна бізнес-ціль або великий модуль. Неможливо зробити за один спринт.

*Приклад: "Система лояльності"*

### Feature (Фіча)

Відчутна функціональна частина епіка.

*Приклад: "Нарахування бонусів за покупку"*

### User Story (Історія)

Конкретний сценарій, який можна реалізувати за 2-3 дні.

*Приклад: "Бачити баланс бонусів у хедері"*

### Task (Завдання)

Робота для розробника, зрозуміла тільки йому.

*Приклад: "Створити таблицю bonuses у БД"*

**Правило:** Замовник оперує Епіками та Фічами. Розробник оперує Історіями та Тасками. Аналітик пов'язує їх між собою.

# Definition of Ready (DoR)

Ми вже говорили про Definition of Done (DoD) — критерії готовності на виході. Але щоб взяти історію в роботу, вона має пройти вхідний контроль — **Definition of Ready (DoR)**.

1

### User Story написана

За форматом "Як... Я хочу... Щоб..."

2

### AC прописані

Критерії приймання чітко визначені

3

### Макет готовий

UI/UX дизайн прикріплений (якщо це UI-задача)

4

### Залежності вирішені

Доступ до API партнера отримано

5

### Задача оцінена

Команда визначила Story Points / Години

Якщо історія не відповідає DoR — її заборонено брати в Спринт. Це захищає команду від простоїв.

# Методи пріоритезації

Найгірший метод пріоритезації — **HiPPO** (Highest Paid Person's Opinion). Це коли роблять те, що сказав начальник, бо він начальник. Інженери мають пропонувати об'єктивні методи.

### Метод RICE

Використовується Intercom та продуктовими компаніями для розрахунку Score (балів).

$$Score_{RICE} = \frac{Reach \times Impact \times Confidence}{Effort}$$

**Reach (Охоплення):** Скількох людей це зачепить? (1000 юзерів)

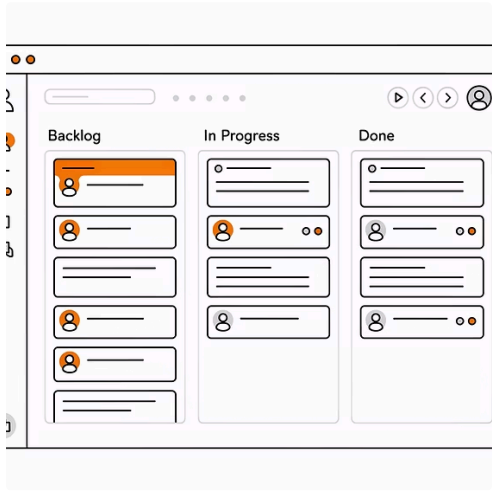
**Impact (Вплив):** Наскільки збільшить продажі? (3 - масивно, 1 - мінімально)

**Confidence (Впевненість):** Наскільки ми впевнені? (100% - є дані, 50% - інтуїція)

**Effort (Зусилля):** Скільки часу займе? (людино-місяців)

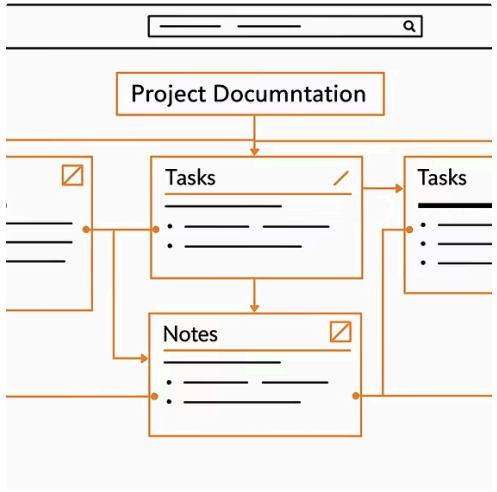
Результат: Робимо задачі з найвищим балом — максимальний ефект за мінімальні зусилля.

# Інструменти документування



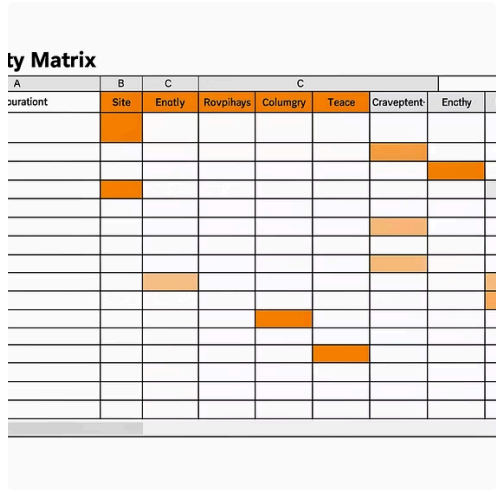

## Jira (Atlassian)

Стандарт де-факто для управління задачами. Дозволяє створювати Беклог, Спринти, вішати лейбли пріоритетів.




## Confluence (Atlassian)

Корпоративна Вікіпедія. Тут зберігаються "довгоживучі" документи: описи Епіків, діаграми C4, інструкції, онбординг.




## Traceability Matrix

Матриця трасування. Показує зв'язок: Вимога → Код → Тест-кейс. Гарантує, що жодна вимога не загубилася.

# Висновки до Лекції

### Вимоги — це паливо

Брудне паливо (погані вимоги) вб'є двигун (проєкт)

### Від ТЗ до Беклогу

User Story фокусується на цінності, AC — на деталях реалізації

### Пріоритезація

RICE — це навичка казати "Ні" хорошим ідеям заради реалізації найкращих

## Наступний крок:

Тепер у нас є список того, що ми хочемо зробити (Беклог). Наступне питання: як це працюватиме зсередини? Як дані будуть рухатися системою?

На наступній лекції ми перейдемо до **Структурного моделювання** і навчимося будувати DFD-діаграми.