Лекція 1

Вступ до інженерії ПЗ та життєвий цикл розробки

Методологія розробки ПЗ: Від хаосу до порядку

Більшість студентів розглядають методології як нудні правила, що заважають писати код. Проте в інженерії ПЗ код — це лише верхівка айсберга.

Методологія — це спосіб управління ризиками та вартістю змін. Вся історія індустрії — це спроба відповісти на одне питання: "Як створити складну систему вчасно, в межах бюджету і так, щоб вона працювала?"

1950-1960-TI

Code and Fix: Дикий захід програмування

Це підхід, з якого починає кожен студент: є задача → пишемо код → запускаємо → падає → фіксимо → запускаємо.

Відсутність планування

Архітектура існує лише в голові розробника

Тестування користувачем

Або самим розробником методом "тику"

Проблема масштабу

Підхід працює для лабораторної роботи на 100 рядків.

У масштабах бізнесу призводить до "Spaghetti code" та повної залежності від конкретних "героїв-розробників".

Криза програмного забезпечення

Наприкінці 1960-х залізо ставало потужнішим, задачі — складнішими (банківські системи, управління польотами). Підхід "Code and Fix" перестав працювати.

Масові провали

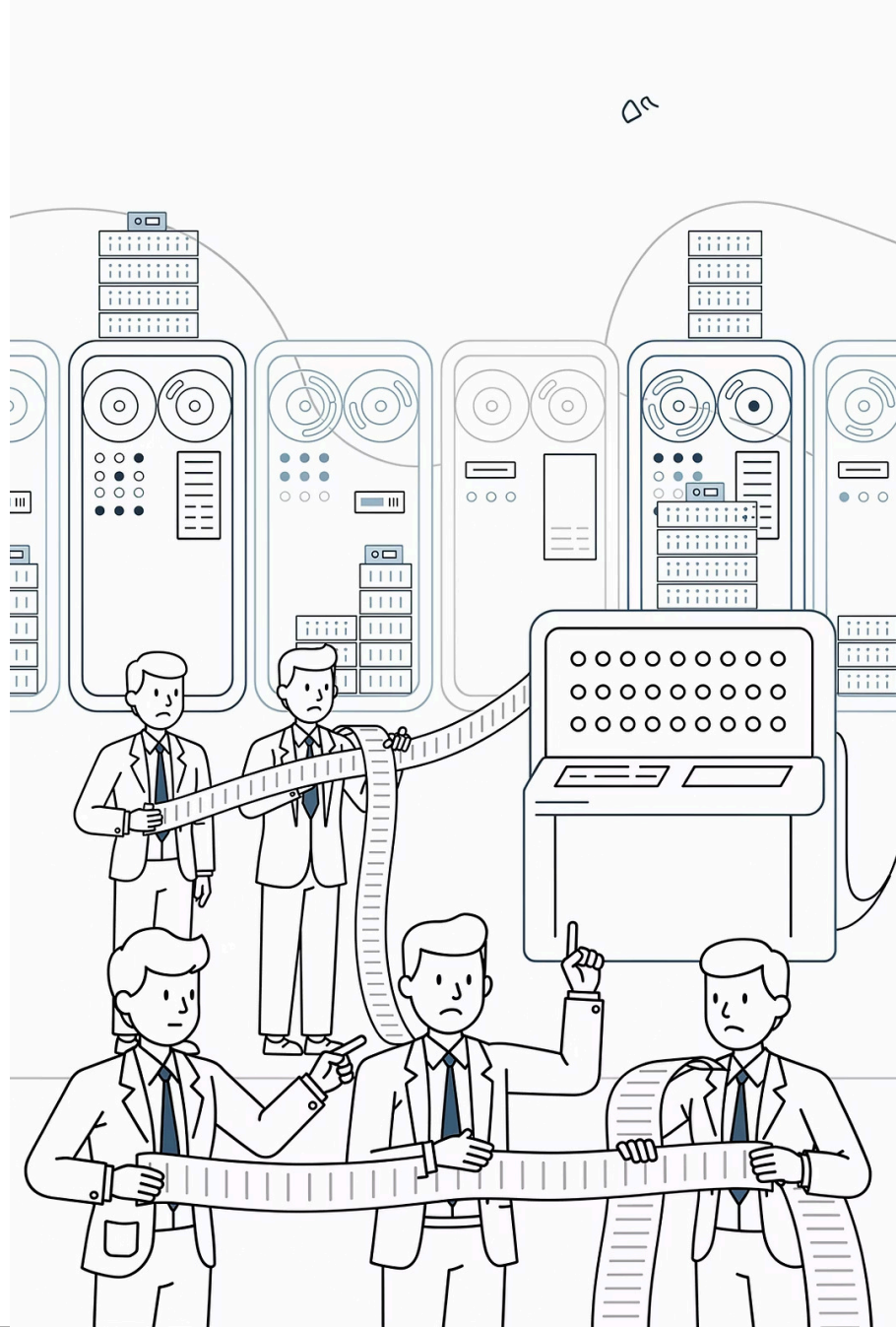
Перевитрати бюджету в рази, зриви дедлайнів на роки

Конференція НАТО 1968

Вперше прозвучав термін Software Engineering

Нова парадигма

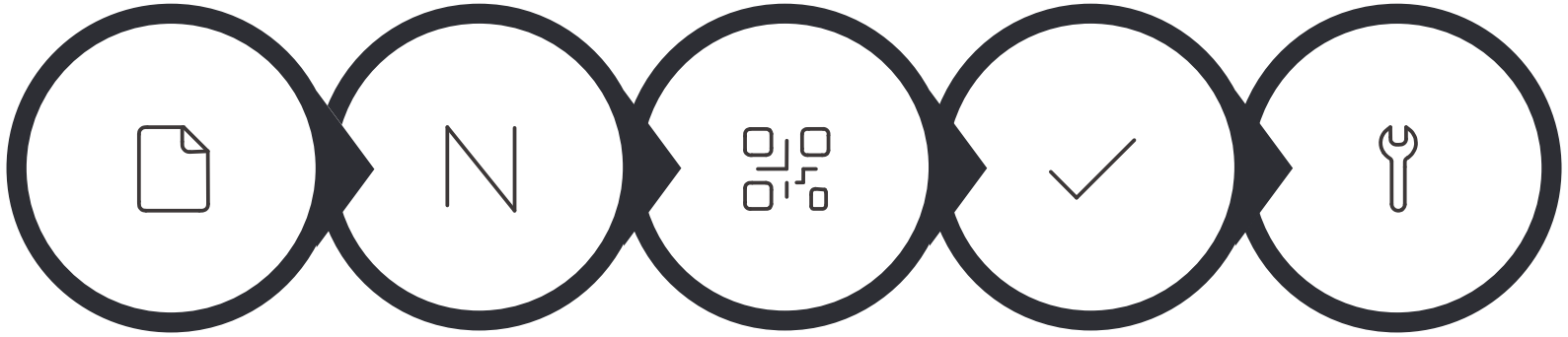
Розробка ПЗ — це інженерна дисципліна, подібна до будівництва мостів



1970-1980-TI

Структурний підхід: Будівельна парадигма

Індустрія звернулася до досвіду цивільної інженерії. Щоб збудувати будинок, ви не починаєте кладку цегли, поки немає затвердженого креслення фундаменту.



Вартість виправлення помилки зростає експоненціально з часом: виправити вимогу на папері коштує $1, архітектуру — $100, баг у продакшні — $10000.

$1

Вимога на папері

$100

Архітектура

$10K

Баг у продакшні

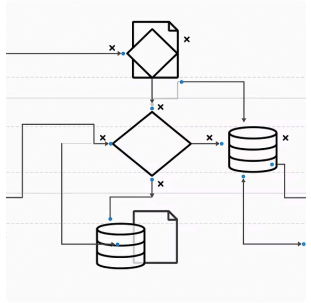
1980-1990-ТІ

Важковаговики та управління ризиками

"Будівельна метафора" виявилася хибною. ПЗ — це не бетон, воно м'яке (software). Вимоги клієнта змінюються швидше, ніж ви встигаєте написати ТЗ. Waterfall виявився надто ригідним.

Спіральна модель (1986)

Фокус на аналізі ризиків на кожному витку спіралі. Підхід для дорогих та небезпечних систем.



RUP (Rational Unified Process)

Важковаговий фреймворк від IBM. Багато документації, діаграм, артефактів. Спроба "описати все".

2001 - СЬОГОДЕННЯ

Agile-революція: Гнучкість понад усе

Ринок змінився. З'явився Інтернет, стартапи, час виходу на ринок став критичним фактором виживання. Ви не можете планувати на рік вперед, бо через рік ваш продукт вже нікому не буде потрібен.

Люди та взаємодія

Важливіші за процеси та інструменти

Працюючий продукт

Важливіший за вичерпну документацію

Співпраця з замовником

Важливіша за узгодження контракту

Готовність до змін

Важливіша за дотримання плану

Це не означає відсутність документації чи планів. Це означає, що адаптивність важливіша за передбачуваність.

Залізний трикутник проєктного менеджменту

Перш ніж порівнювати методології, згадаємо класичний трикутник: Scope (Обсяг), Time (Час), Cost (Вартість). Якість знаходиться посередині.

> Ви не можете зафіксувати жорстко всі три параметри одночасно. Якщо ви хочете зафіксувати все — якість гарантовано впаде.

Фундаментальна різниця між Waterfall та Agile полягає в тому, що саме ми фіксуємо, а що залишаємо змінним.

Каскадна модель: Передбачуваність понад усе

Класичний, лінійний підхід, успадкований від інженерії матеріального світу. Процес розбивається на послідовні фази. Перехід на наступну фазу можливий лише після повного завершення попередньої.

1

Вимоги

Детальне ТЗ на 100 сторінок

2

Проєктування

Архітектура всієї системи

3

Реалізація

Кодування місяцями

4

Тестування

Перевірка всього разом

5

Підтримка

Що фіксуємо

Scope (Обсяг): ЗАФІКСОВАНИЙ. Ми точно знаємо, що будуємо.

Time & Cost: ОЦІНОЧНІ. Намагаємось вгадати час реалізації.

Ідеально для

- Типових проєктів з фіксованими вимогами
- Медичне обладнання, авіоніка
- Атомні станції, будівництво
- Де ціна помилки критична

Недоліки Waterfall: Чому всі тікають в Agile

1

Ефект тунелю

Замовник бачить результат тільки в кінці. Якщо ви 6 місяців будували "не те", ви дізнаєтесь про це на фініші.

2

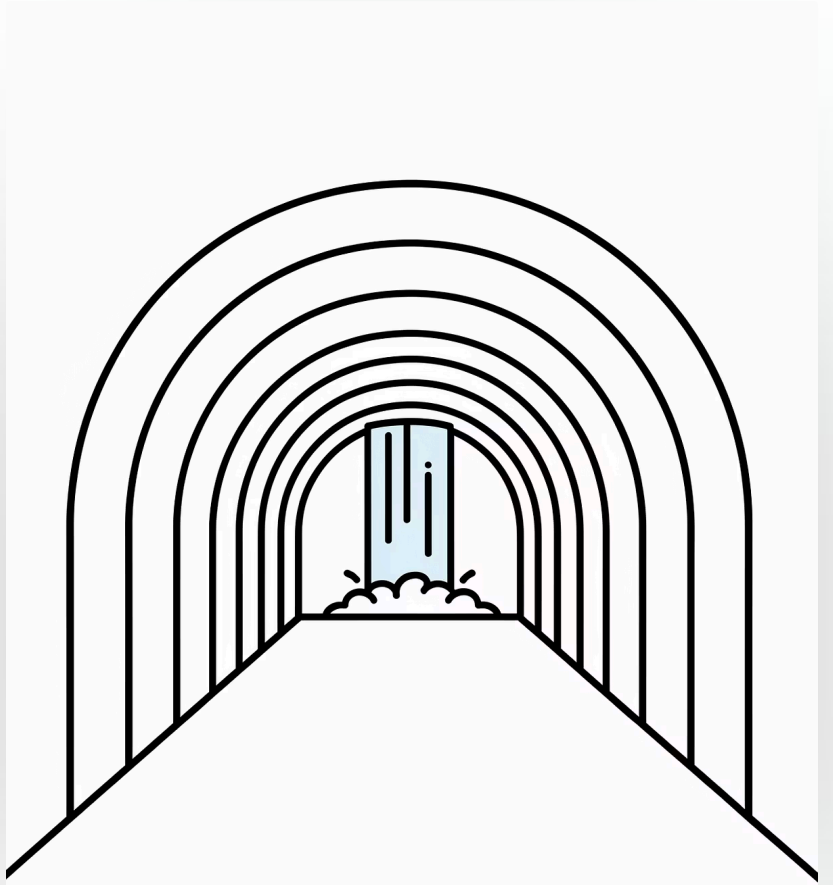
Ілюзія визначеності

На початку проєкту ми знаємо про нього найменше. Waterfall змушує приймати найважливіші рішення саме тоді, коли ми найменше розуміємо проблему.

3

Опір змінам

Будь-яка зміна вимог вимагає переписування документації та перегляду бюджету. Зміни стають дорогими та болючими.



Гнучкі методології: Адаптивність понад усе

Agile — це сімейство методологій (Scrum, Kanban, XP), об'єднаних спільними цінностями. Ітеративний та інкрементальний підхід.

Ми не будуємо весь продукт одразу. Розбиваємо його на маленькі шматочки і реалізуємо короткими циклами (спринтами — 2-4 тижні). В кінці кожного циклу маємо готовий до використання інкремент.

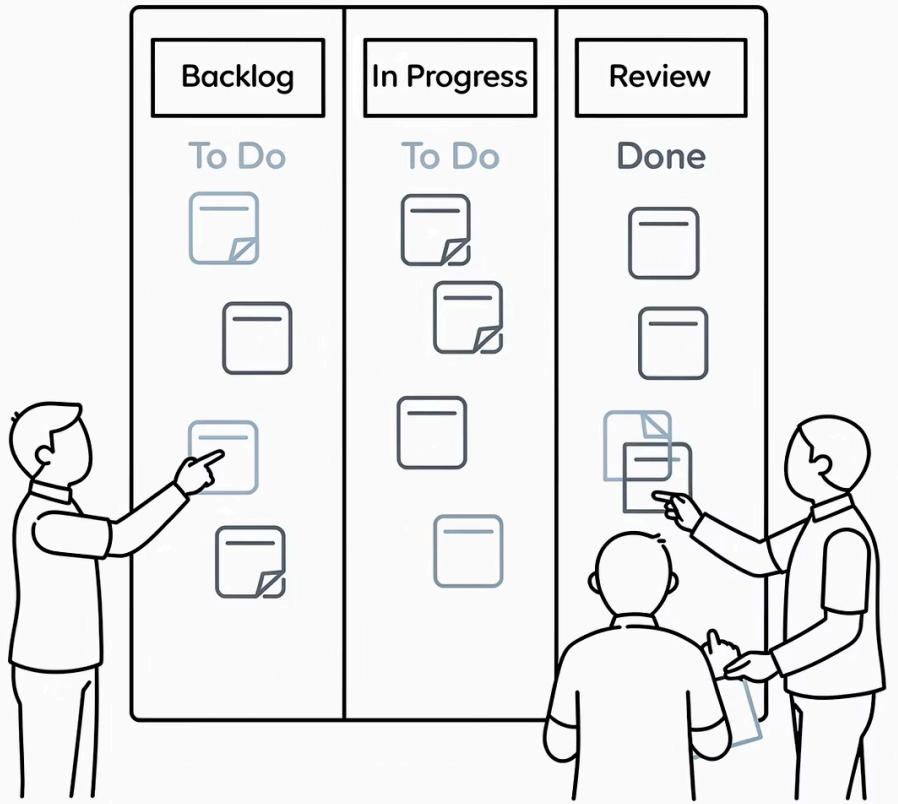
Що фіксуємо

Time & Cost: ЗАФІКСОВАНІ. Фіксована команда і час спринту (2 тижні).

Scope: ЗМІННИЙ. Робимо найважливіші функції, решту переносимо.

Філософія

"Fail fast, fail cheap" — помиляйся швидко, помиляйся дешево. Якщо гіпотеза хибна, дізнаємось через 2 тижні, а не через рік.

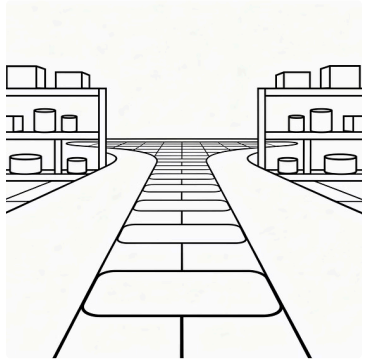


Waterfall vs Agile: Порівняльна таблиця

Критерій	Waterfall	Agile
Підхід	Послідовний (Sequential)	Ітеративний (Iterative)
Вимоги	Чітко визначені на старті	Еволюціонують в процесі
Участь замовника	Висока на старті, низька в процесі	Постійна протягом проєкту
Тестування	В кінці розробки	Постійно, паралельно
Результат	Готовий продукт в кінці	Інкремент кожні 2-4 тижні
Ставлення до змін	Зміни — це зло і витрати	Зміни — це можливість
Документація	Формальна, вичерпна	Достатня, "Just-in-time"

Як обрати методологію? Cynefin Framework

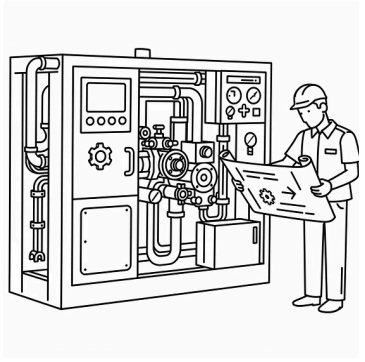
Для прийняття рішення використовують модель Cynefin, яка ділить системи на типи залежно від складності та передбачуваності.



Прості / Зрозумілі

Точно знаєте, що робити (лендінг на шаблоні)

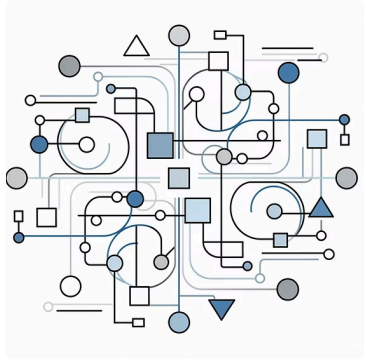
Рекомендація: Waterfall



Складні

Вимагають експертизи, але передбачувані (міграція БД)

Рекомендація: Waterfall або Гібрид



Заплутані / Комплексні

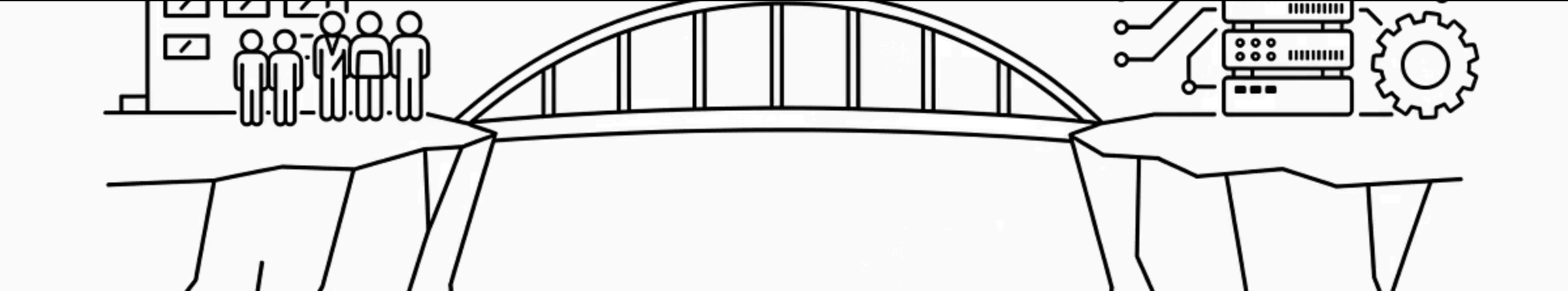
Не знаєте реакції користувачів (стартап, новий сервіс)

Рекомендація: Agile

Хаотичні

Кризова ситуація (сервер впав, все горить)

Рекомендація: Діяти негайно (Kanban)



Хто перетворює хаос на порядок?

Методологія — це лише інструкція. Щоб вона запрацювала, потрібні конкретні ролі, які відповідають за розуміння проблеми та пошук технічного рішення.

> Бізнес знає, скільки грошей він хоче заробити. Розробник знає, як написати цикл for. Між ними лежить прірва.

Системний аналітик

Відповідає за питання "ЩО ми будуємо?"

Архітектор

Відповідає за питання "ЯК ми це будуємо?"

Системний аналітик: Перекладач з людської на айтішну

Бізнесмени розмовляють мовою прибутків та KPI. Програмісти — мовою об'єктів та API. Якщо звести їх напряму, вийде гра "зіпсований телефон". Аналітик виступає професійним перекладачем.

01

Elicitation (Виявлення вимог)

"Витягування" інформації зі стейкхолдерів. Клієнт часто сам не знає, чого хоче.

02

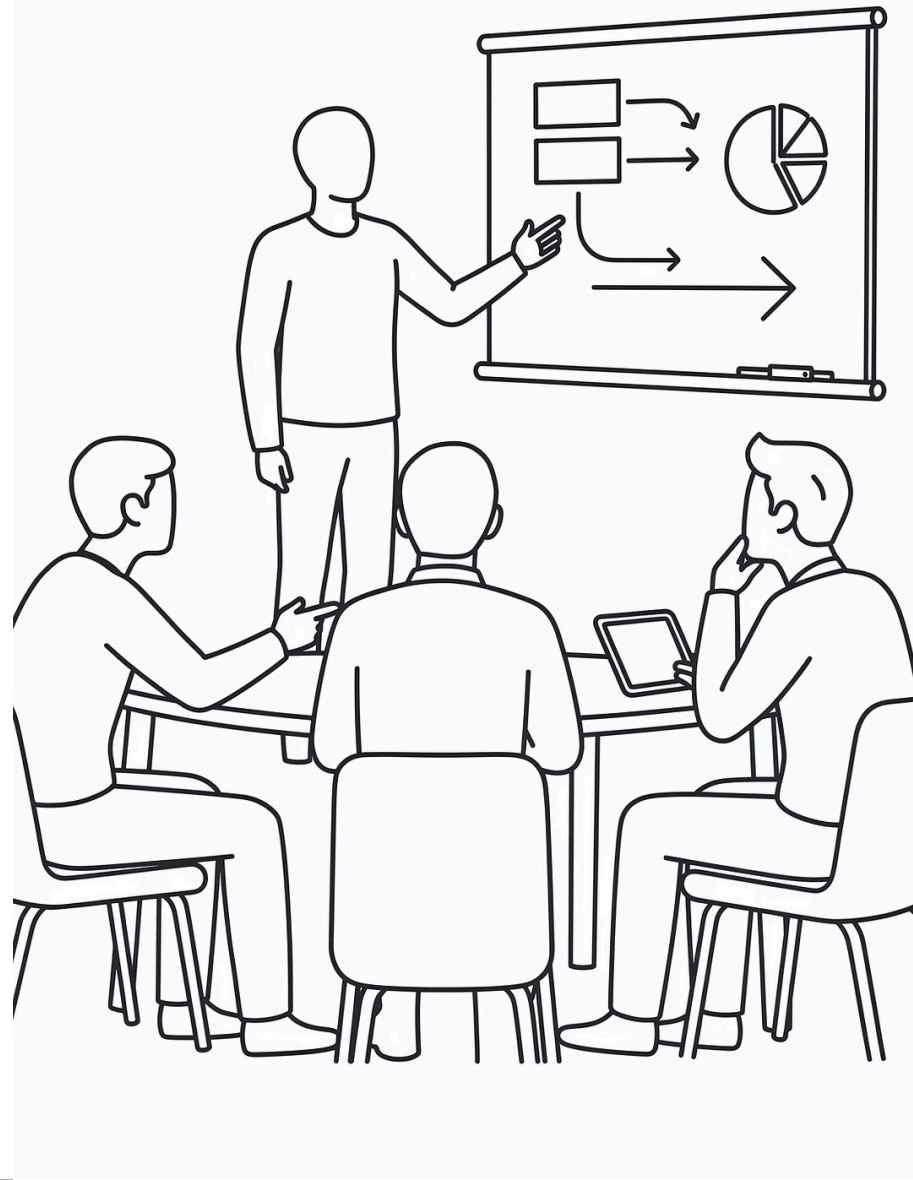
Analysis (Аналіз)

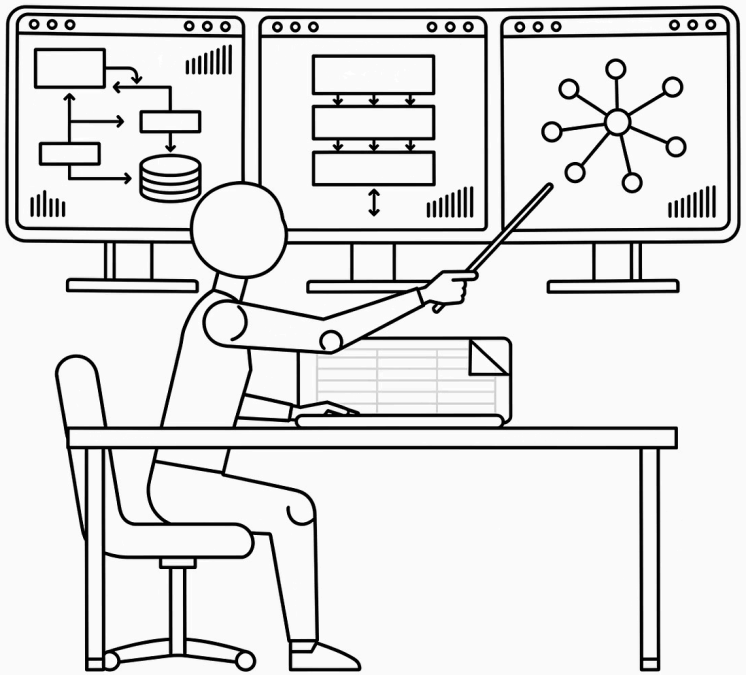
Пошук суперечностей. Маркетинг хоче акцію для всіх, а Юристи кажуть — незаконно для осіб до 18.

03

Documentation (Специфікація)

Перетворення розмов на чіткі документи (User Stories, Use Cases), які розробник зрозуміє однозначно.





Програмний архітектор: Головний інженер проєкту

Якщо аналітик каже "Нам потрібен будинок з 3 спальнями", то архітектор вирішує: "Будуємо з цегли чи дерева? Який фундамент витримає 3 поверхи?"

Архітектор приймає High-Level Technical Decisions — рішення, які найважче і найдорожче змінити згодом.

Вибір технологічного стеку

Java чи Python? SQL чи NoSQL? React чи Angular? Це питання відповідності вимогам, а не моди.

Проєктування структури

Розбиття системи на модулі, мікросервіси чи шари. Component Design.

Нефункціональні вимоги

Система має витримати 1 млн користувачів одночасно. Security, Scalability, Performance.

Взаємодія: Аналітик vs Архітектор vs Розробник

На практиці ці ролі знаходяться у стані конструктивного конфлікту, балансуючи між бізнес-потребами та технічними можливостями.

Аналітик (Бізнес)

"Клієнту треба, щоб пошук працював миттєво і шукав навіть за помилками в словах!"

Архітектор (Технології)

"Щоб це працювало миттєво, нам треба ElasticSearch і 3 додаткові сервери. Це $5000/міс. Бізнес згоден?"

Розробник (Реалізація)

"Дайте мені чітке API, і я це закоджу. Але не змінюйте вимоги кожні 5 хвилин!"

Важливо для кар'єри: У невеликих компаніях роль Архітектора виконує Senior Developer. Роль Аналітика ділять PM та Ліди. На великих Enterprise-проєктах це окремі люди, бо ціна помилки — мільйони.

Артефакти проєктування: Креслення для коду

В IT артефакт — це будь-який матеріальний результат роботи команди, створений у процесі розробки ПЗ. Код складно читати, тому ми створюємо проєктні артефакти — "креслення", за якими будується "будинок".

Текстові артефакти

- Візія проєкту (Vision)
- Реєстр вимог (Backlog)
- Сценарії користування (Use Cases)
- API документація (Swagger)

Графічні артефакти

- Діаграми (UML, C4, DFD)
- Макети інтерфейсу (Wireframes)
- Прототипи (Figma)

> Результат роботи Аналітика = Вимоги. Результат роботи Архітектора = Архітектурна документація. Це "паливо" для розробників.

Еволюція моделювання: Від паперових монстрів до живих схем

1 SADT, IDEF0, ГОСТ

Епоха Waterfall. Гігантські формалізовані схеми. Важко створювати і підтримувати. "Мертва" документація.

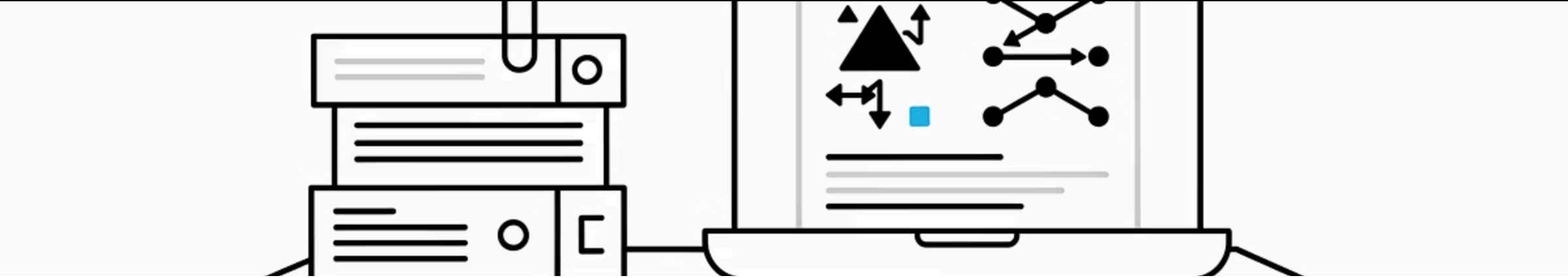
2 UML (1990-ті)

Золотий стандарт індустрії. 14 типів діаграм. Сьогодні використовуємо як ескіз, а не детальне креслення.

3 Agile Modeling, C4

Сучасний прагматизм. Діаграми для розмови. Documentation as Code. Зберігаються в Git разом з кодом.

Ми не використовуємо UML так, як 20 років тому (як креслення для генерації коду). Ми використовуємо UML як ескіз для обговорення складних моментів.



Принцип "Just Enough": Необхідно і достатньо

В Agile існує міф: "Ми не пишемо документацію". Правда: "Ми пишемо документацію, яка має цінність, і робимо це в останній відповідальний момент".

Bus Factor

Якщо завтра головного архітектора "зіб'є автобус", чи зможе команда продовжити роботу?

Знання тільки в голові

Команда не зможе продовжити без ключової людини

Актуальні артефакти

Схеми та вікі дозволяють команді працювати далі

Кожен артефакт відповідає на конкретне питання бізнесу: Vision & Scope (Навіщо?), User Stories (Чого хоче користувач?), DFD (Куди йдуть дані?), UML Class (Структура БД?), C4 (З яких серверів складається система?).

Висновки: Готові до практики

1 **Agile переміг у розробці продуктів**

Але Waterfall актуальний для типових задач з фіксованими вимогами

2 **Аналітик шукає проблему, Архітектор — рішення**

"Що?" vs "Як?" — дві сторони однієї медалі

3 **Артефакти — це комунікація**

Найкращий артефакт — той, який легко читати і легко оновлювати

4 **Відмовляємось від бюрократії**

Використовуємо UML-ескізи та модель C4 замість складних моделей

Наступний крок: На першій лабораторній роботі ви сформуєте Vision свого навчального проєкту — документ, який відповідає на питання: Яку проблему ми вирішуємо? Для кого? Чим ми кращі за конкурентів?