Лекція 3

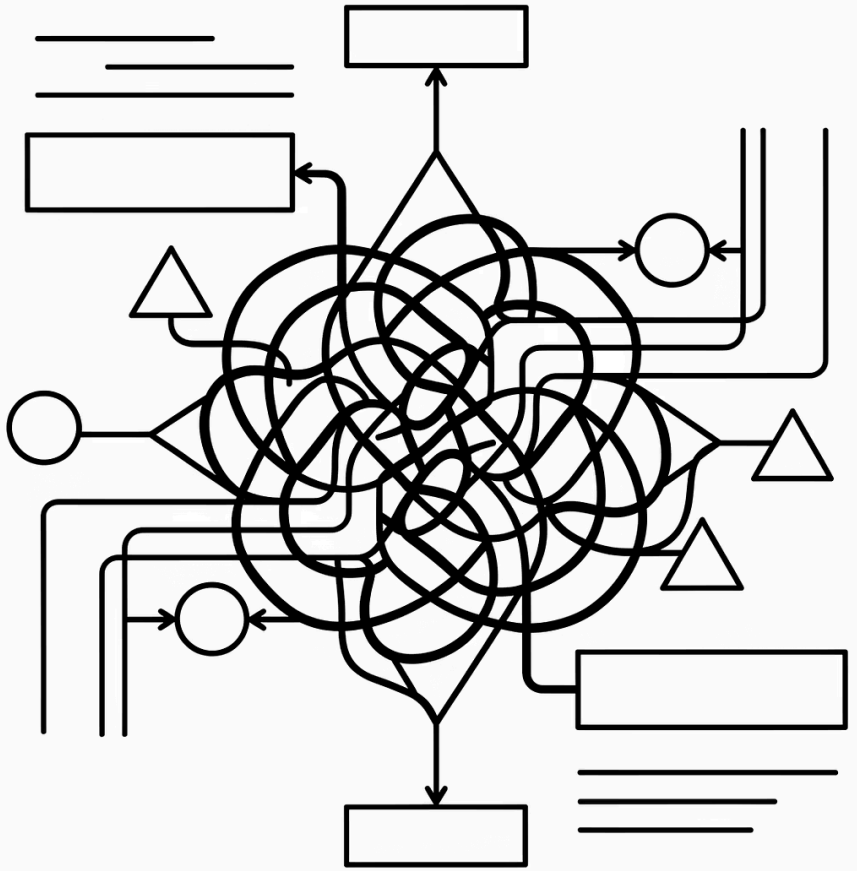
Структурне моделювання процесів

Проблема "Спагетті-мислення"

Перш ніж писати код (ООП, класи, мікросервіси), нам потрібно зрозуміти, що система робить. Багато початківців одразу кидаються проєктувати базу даних або малювати інтерфейс. Це помилка.

Якщо ви не розумієте, як інформація рухається вашою системою, ви створите "спагетті-код" і заплутану архітектуру. Структурний аналіз (Structured Analysis) — це методологія, яка дозволяє перетворити хаотичну, складну проблему на набір чітких, ієрархічно впорядкованих функцій.

> Це "рентген" вашої майбутньої системи. Він відповідає на питання: «Які перетворення даних відбуваються всередині?»



Філософія: "Divide and Conquer"

РОЗДІЛЯЙ І ВОЛОДАРЮЙ

В основі структурного аналізу лежить один головний інженерний принцип — Декомпозиція. Людина не здатна тримати в голові одночасно більше 7±2 об'єктів (закон Міллера). Якщо ви спробуєте описати роботу банку однією схемою, ви зійдете з розуму.

Принцип абстрагування

Виділяємо лише суттєві аспекти системи, ігноруючи деталі на поточному рівні. Як карта країни показує міста точками, без світлофорів.

Ієрархічна впорядкованість

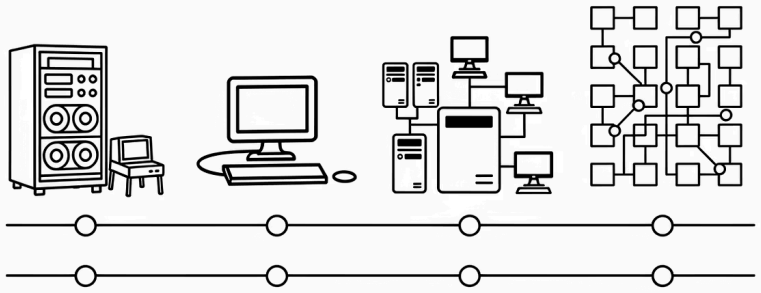
Система будується зверху вниз (Top-Down). Від "Чорної скриньки" до деталей підсистем через рівні декомпозиції.

Принцип формалізації

Використання суворих графічних нотацій. Словесний опис неприпустимий — тільки діаграми з чітким значенням.

Принцип "Чорної скриньки"

На кожному рівні концентруємося на тому, ЩО робить функція (вхід → перетворення → вихід), а не ЯК.



Історичний контекст: SADT vs Сучасність

70-80-ті роки

Структурний аналіз був домінуючим (SADT, IDEF0). Програми були процедурними (C, Pascal), підхід "функція викликає функцію" ідеально лягав на код.

Повернення у сучасність

З приходом ООП у 90-х структурний аналіз оголосили "застарілим". Але сьогодні він повертається!

- **Мікросервіси:** Архітектура мікросервісів — це чистий структурний підхід
- **Serverless (FaaS):** AWS Lambda — реалізація принципу декомпозиції процесів
- **Data Engineering:** ETL-пайплайни вимагають чіткого розуміння потоків даних

Знання структурного аналізу сьогодні відрізняє Architect від звичайного розробника.

Загальна схема проведення структурного аналізу

01

Контекст

Визначити межі системи. Хто дає дані, а хто забирає результат? Окреслюємо кордони пісочниці.

02

Декомпозиція

Розбити систему на 3-7 великих функціональних блоків.

03

Аналіз потоків

Намалювати стрілки: які саме документи/JSON-и літають між цими блоками.

04

Деталізація

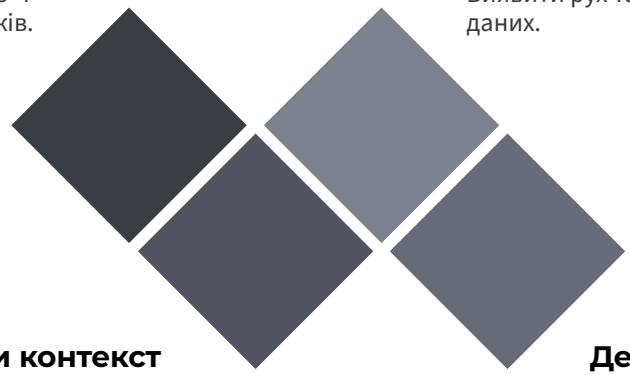
Якщо блок занадто складний — "провалюємось" всередину і повторюємо процес рекурсивно.

Декомпозиція

Розбити систему на 3–7 блоків.

Аналіз потоків

Виявити рух та передачу даних.



Визначити контекст

Задати межі та джерела даних.

Деталізація

Рекурсивно уточнити кожен блок.

Процес, який ви будете виконувати на лабораторних, складається з чотирьох послідовних кроків, кожен з яких поглиблює розуміння системи.

Що таке DFD і чому це не Блок-схема?

⚠ НАЙПОШИРЕНІША ПОМИЛКА

Студенти часто плутають DFD (Data Flow Diagram) та Flowchart (Блок-схему алгоритму). Це принципово різні інструменти з різними цілями.

Блок-схема (Flowchart)

Показує **порядок дій** (Control Flow). Відповідає на питання: "Що робити далі? Якщо А, то Б, інакше В".

- Є цикли та умови if/else
- Показує послідовність у часі
- Інструкція для кухаря: порізати → посмажити → подати

DFD (Data Flow Diagram)

Показує **рух інформації** (Data Flow). Відповідає на питання: "Звідки прийшли дані, як вони змінилися і куди лягли?"

- Немає часу, немає циклів
- Процеси можуть відбуватися одночасно
- Схема водопроводу: звідки тече вода, в які фільтри потрапляє

Основні елементи DFD

Нотація Gane-Sarson

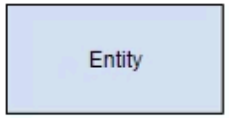
Модель складається всього з 4 елементів. Кожен має своє призначення та правила найменування.

1

Зовнішня сутність (External Entity)

Квадрат або прямокутник. Джерело або споживач даних за межами вашої системи.

Приклади: "Клієнт" (людина), "Банківський API" (чужа система), "Адміністратор" (роль).



2

Потік даних (Data Flow)

Стрілка, що показує шлях переміщення пакета даних.

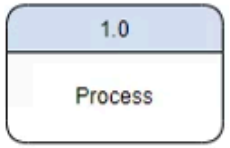
Правило: Іменник. Підписуйте, що саме летить. Добре: "Заявка на кредит", "ID клієнта".

3

Процес (Process)

Прямокутник із заокругленими кутами. Перетворює вхідні дані на вихідні — "м'ясорубка" для інформації.

Правило: Завжди Дієслово + Іменник. Добре: "Перевірити замовлення", "Розрахувати зарплату".



4

Сховище даних (Data Store)

Прямокутник з відкритим правим боком. Місце, де дані зберігаються (таблиця БД, файл, черга).

Правило: Іменник у множині. Приклад: "Клієнти", "Товари", "Замовлення".

Правила з'єднань: Синтаксис DFD

У DFD є "закони фізики", порушувати які не можна. Всі зв'язки мають проходити через Процес.

❌ Entity ↔️ Entity

Зовнішні сутності не спілкуються напряму через вашу діаграму. Якщо Клієнт говорить з Банком — це не ваша система.

❌ Store ↔️ Store

Одне сховище не може передати дані в інше саме по собі. Потрібен процес ("Копіювання", "Архівування").

❌ Store ↔️ Entity

Користувач не лізе руками в базу даних. Потрібен процес ("Інтерфейс", "Читання"), який дістане дані.

Єдиний правильний шлях: Entity → Process → Store (Збереження даних) • Store → Process → Entity (Відображення даних) • Process → Process (Обробка).

Типові логічні помилки

ЩО ШУКАТИ НА ЛАБОРАТОРНИХ

Під час перевірки робіт я буду шукати у ваших схемах ці три критичні помилки. Кожна з них порушує логіку руху даних.

Чорна діра (Black Hole)

Процес, який має тільки вхідні стрілки, але не має вихідних. Дані зникають.

Приклад: Процес "Розрахувати податок" отримав зарплату, але нікуди не віддав результат.

Виняток: Такого не буває. Навіть якщо це запис у лог, має бути стрілка в сховище "Логи".

Диво (Miracle)

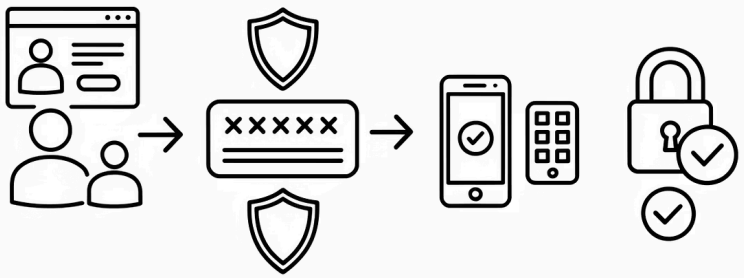
Процес, який генерує дані з повітря (тільки вихідні стрілки).

Приклад: Процес "Сформувати рахунок" видає PDF, але на вхід йому нічого не прийшло. Звідки він знає суму і ім'я клієнта?

Сіра діра (Grey Hole)

Найскладніша помилка. Вхідні дані недостатні для отримання вихідних.

Приклад: На вході "Дата народження", а на виході "Номер паспорта". Ви не можете отримати паспорт з дати народження. Вхід і вихід не узгоджені.



Приклад побудови: Процес авторизації

Розглянемо простий кейс, щоб побачити, як абстрактне слово "Логін" перетворюється на інженерну схему руху даних.

1

Користувач → Процес

Стрілка від Користувача до "Перевірити пароль" (підпис: "Логін/Пароль")

2

Процес → Сховище

Стрілка від "Перевірити пароль" до БД "Користувачі" (підпис: "Запит по логіну")

3

Сховище → Процес

Стрілка від БД "Користувачі" до "Перевірити пароль" (підпис: "Хеш збереженого пароля")

4

Процес → Користувач

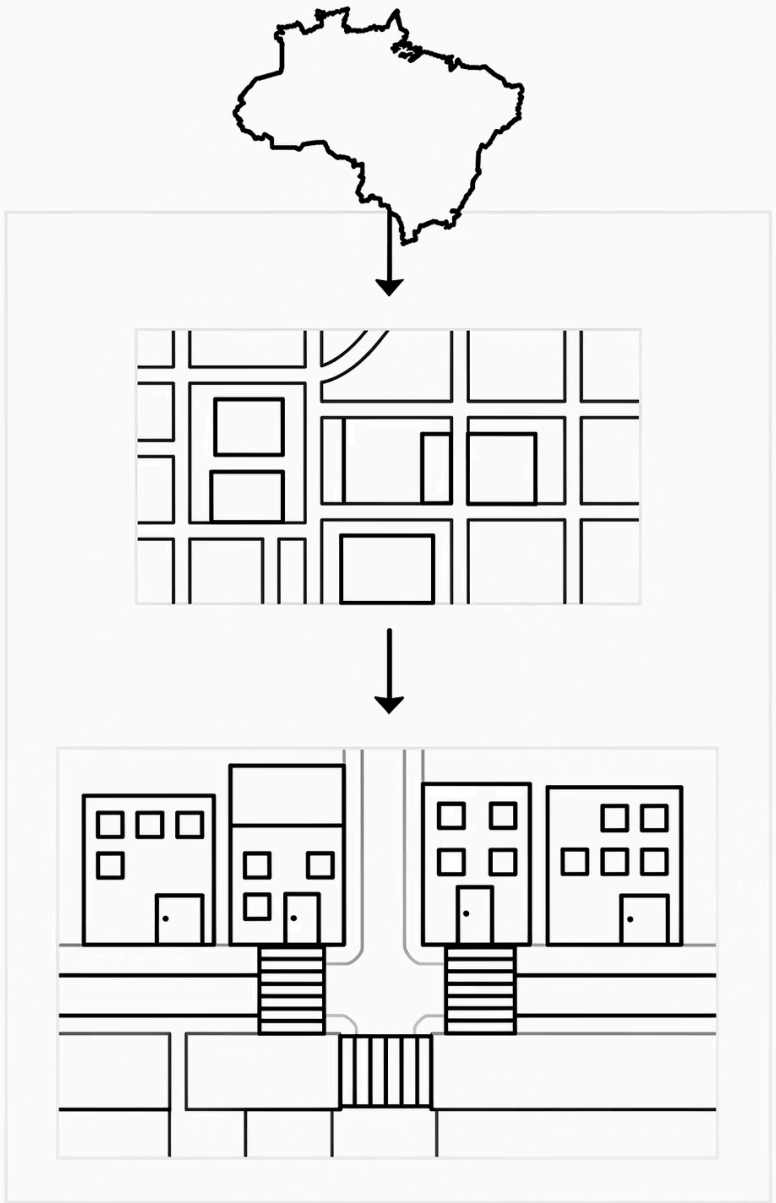
Стрілка від "Перевірити пароль" назад до Користувача (підпис: "Токен доступу" або "Помилка")

Принцип "Google Maps"

Уявіть, що ви відкриваєте Google Maps. Спочатку ви бачите країну (загальні обриси). Наближаєте (Zoom In) — бачите міста і траси. Наближаєте ще — бачите вулиці та будинки.

DFD працює так само. Ми не можемо намалювати всю систему на одному аркуші А4 — вийде нечитабельна павутина. Тому ми створюємо набір діаграм, вкладених одна в одну, як матрьошки.

> Цей процес називається **функціональною декомпозицією**.



Рівень 0: Контекстна діаграма

ПОГЛЯД З КОСМОСУ

Це найвищий рівень абстракції. Найважливіша діаграма для узгодження меж проєкту із замовником (Project Scope).

Характеристики

- **Тільки ОДИН процес:** По центру діаграми один великий прямокутник або коло, яке символізує всю вашу систему
- **Немає сховищ даних:** Внутрішні бази даних сховані всередині системи
- **Зовнішні сутності:** Показуємо всіх, хто взаємодіє з системою
- **Потоки:** Тільки входи та виходи на кордоні системи

Приклад діаграми

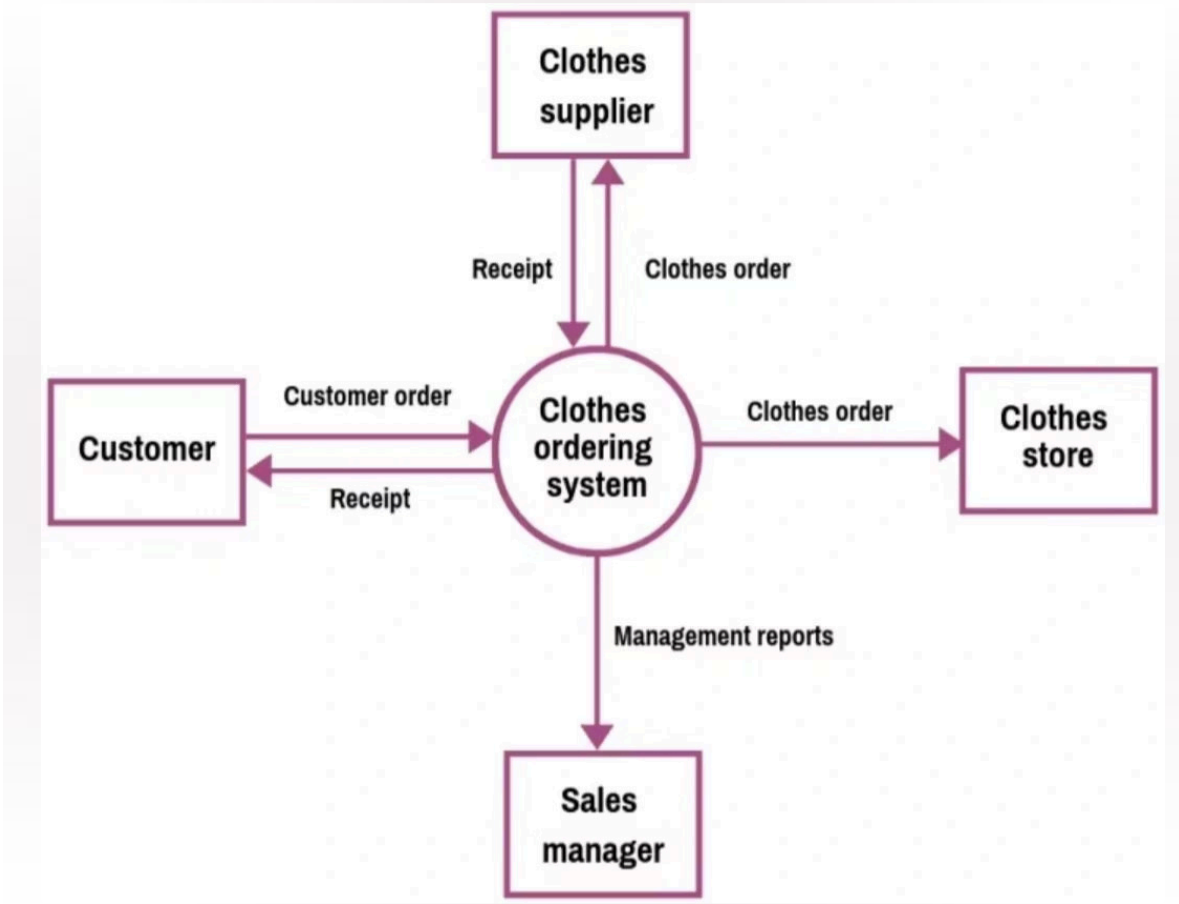
По центру процес "Система замовлення одягу".

Зліва сутність "Клієнт" надсилає "Замовлення" і отримує "Чек".

Справа сутність "Магазин" отримує "Замовлення на одяг".

Зверху "Постачальник" отримує "Замовлення на одяг" та надсилає "Чек"

Знизу "Менеджер" отримує "Звіт"



Рівень 1: Діаграма першого рівня

Ми беремо "лупу" і дивимось всередину процесу "0". Розбиваємо "Чорну скриньку" на основні підсистеми.

Управління каталогом

Процес №1. Додавання, редагування, пошук товарів.

Обробка кошика

Процес №2. Додавання товарів, розрахунок суми.

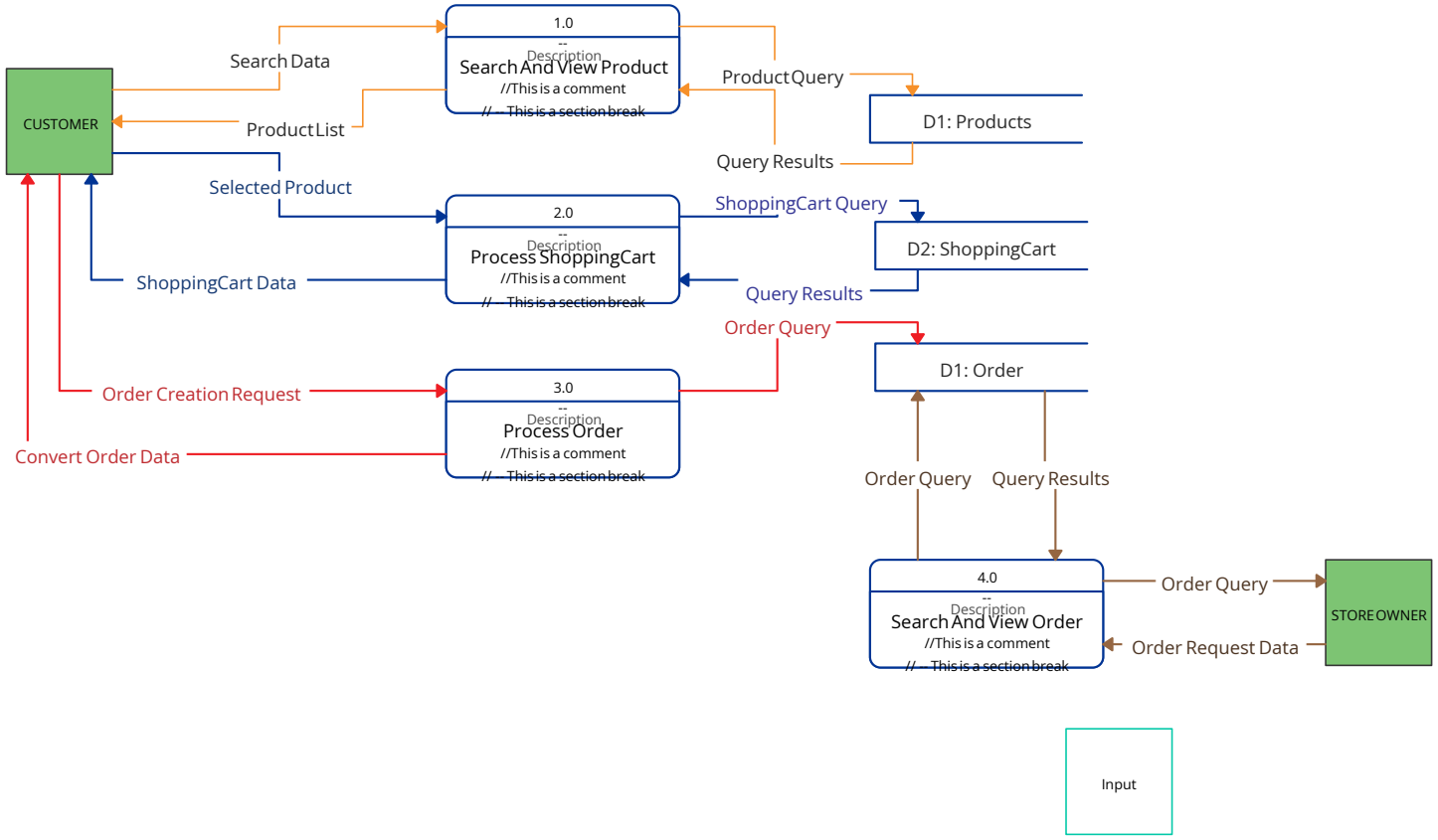
Прийом оплати

Процес №3. Валідація, запит до банку, підтвердження.

Оформлення доставки

Процес №4. Вибір адреси, розрахунок вартості.

Зазвичай 3-7 основних функціональних блоків. Тут вперше з'являються бази даних (наприклад, "Товари", "Клієнти").



Рівень 2: Діаграма другого рівня

Якщо якийсь процес на Рівні 1 все ще занадто складний, ми декомпозуємо його далі. Це рівень конкретних логічних кроків, близьких до функцій у коді.

3.1 Валідувати картку

Перевірка формату номера, терміну дії, CVV-коду

3.2 Надіслати запит у шлюз

Формування та відправка запиту до платіжної системи

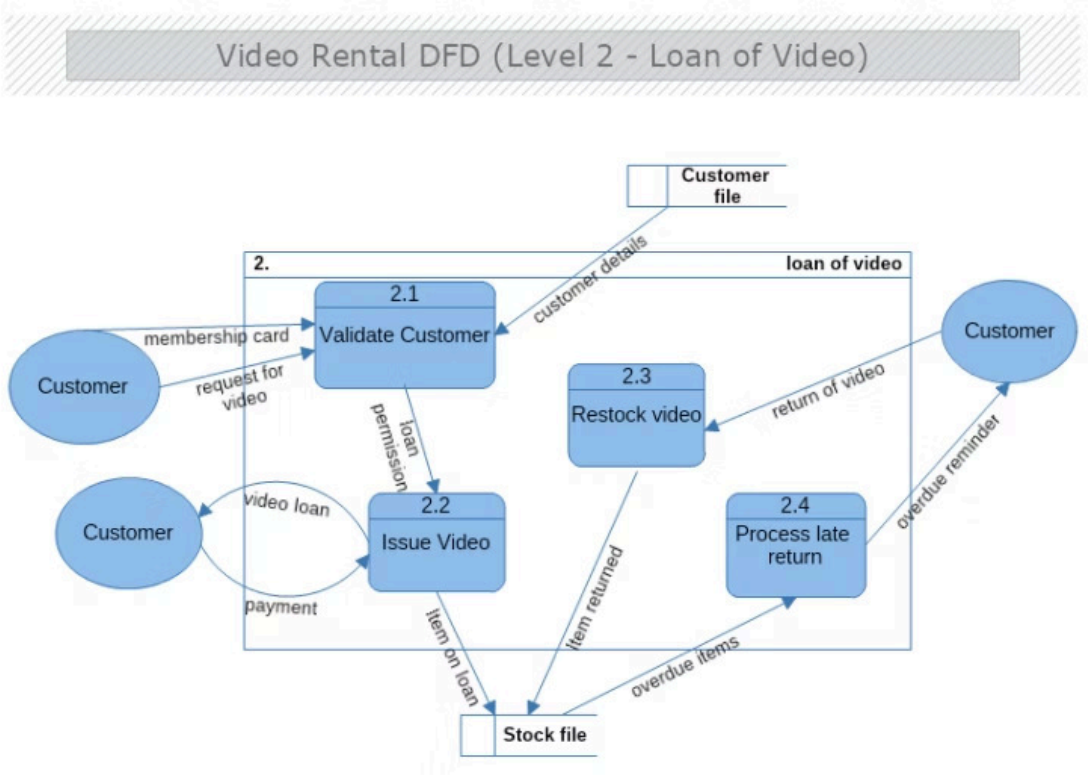
3.3 Оновити статус

Зміна статусу замовлення в базі даних

3.4 Надіслати чек

Генерація та відправка електронного чека на Email

Нумерація: Використовується крапка. Якщо ми розкриваємо процес №3, то його підпроцеси будуть: 3.1, 3.2, 3.3.



Коли зупинитися?

"Чи треба малювати рівень 3, 4, 5?" Зазвичай Рівня 2 достатньо.

Ми зупиняємось, коли процес стає елементарним (Primitive Process).

1 Одне дієслово

Його можна описати одним дієсловом ("Зберегти", "Розрахувати", "Видалити")

Половина сторінки

Його логіка поміщається на половину сторінки тексту (міні-специфікація)

Без алгоритмів

Подальший поділ перетворює діаграму на блок-схему алгоритму (з'являються if/else), а DFD цього не любить

Зовнішні сутності (External Entities)

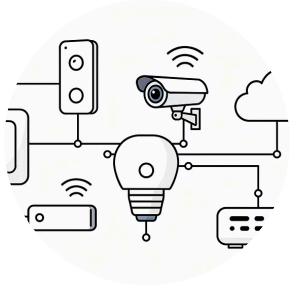
Це джерела даних (Source) або їх споживачі (Sink). Головне правило: Це об'єкти, які ми НЕ проєктуємо і НЕ програмуємо. Ми лише будуємо з ними інтерфейс взаємодії.

Люди (Ролі)

Приклади: "Клієнт", "Адміністратор", "Менеджер складу". Використовуйте назву ролі, а не конкретні імена.

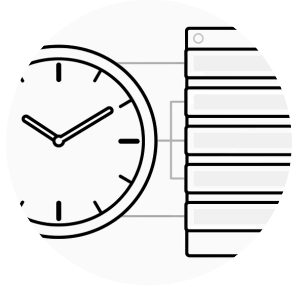
Інші IT-системи

Приклади: "Платіжний шлюз (LiqPay)", "Google Maps API", "1С:Бухгалтерія". Чорні скриньки з чітким інтерфейсом.



Пристрої (IoT / Hardware)

Приклади: "Термодатчик", "Сканер штрих-кодів", "GPS-модуль". Генерують потоки даних автоматично.



Час (Time)

Специфічна сутність. Якщо процес запускається за розкладом, джерелом даних виступає "Таймер" або "Планувальник (Cron)".

> **Як розпізнати сутність?** Задайте питання: "Чи можу я змінити код цього об'єкта?" Якщо Ні — це Зовнішня сутність. Якщо Так — це частина вашої системи (Процес).

Сховища даних (Data Stores)

ПАМ'ЯТЬ СИСТЕМИ

Це місце, де дані зупиняються (Data at Rest). Студенти часто думають, що Data Store = SQL База Даних. Це не так. У DFD сховищем є будь-що, що зберігає стан.

Фізична реалізація

- **База даних:** PostgreSQL, MySQL
- **Файл:** XML, JSON, CSV на диску
- **Кеш:** Redis, Memcached — тимчасове сховище
- **Черга повідомлень:** RabbitMQ, Kafka — буфер
- **Паперовий журнал:** Якщо система не повністю автоматизована

Правила роботи

Тільки пасивність: Сховище не може саме ініціювати дію. Це має зробити Процес моніторингу.

Читання/Запис:

- Стрілка до сховища = Запис (Create/Update)
- Стрілка від сховища = Читання (Read)
- Двостороння стрілка = Читання з оновленням (Modify)

Практичний кейс: "Кошик покупця"

Розглянемо типову помилку при моделюванні процесу додавання товару в кошик.

❌ Варіант А (Помилковий)

Сутність "Клієнт" → стрілка "Товар" → Сховище "Кошик".

Чому погано: Клієнт не має прямого доступу до таблиці БД. Він не пише SQL-запити. Порушено правило з'єднань.

✅ Варіант Б (Правильний)

Сутність "Клієнт" → стрілка "Запит на додавання (ID товару)" → Процес "Додати в кошик".

Процес "Додати в кошик" → стрілка "Дані товару" → Сховище "Кошик (Session/Redis)".

Як не заплутатися? Чек-ліст перевірки

Коли ви малюватимете DFD на лабораторній, перевірте себе за цими критеріями. Це допоможе уникнути найпоширеніших помилок.

1

Перевірка сутностей

Чи всі сутності дійсно зовнішні? Якщо ви намалювали сутність "База даних", ви помилилися. База даних — це внутрішнє Сховище (Store), а не Сутність (Entity), бо ви її проєктуєте.

2

Зворотний зв'язок

Чи не забули ви "зворотний зв'язок"? Якщо Клієнт надсилає "Замовлення", він має отримати "Номер замовлення" або "Помилку". Сутності рідко працюють в одні ворота (крім датчиків).

3

Найменування сховищ

Чи названі сховища іменниками у множині? "Клієнти", "Замовлення", "Налаштування". Це допомагає відрізняти їх від процесів.

Висновки до Лекції 3

Ми завершили розгляд структурного моделювання. Тепер ви знаєте, як побудувати "рентгенівський знімок" системи за допомогою DFD.

Декомпозиція

Розбили складне на просте

Data Flow

З'єднали процеси потоками

Пам'ять

Визначили пам'ять (Сховища)

Межі

Визначили межі (Сутності)

Цей підхід ідеальний для проєктування Back-end логіки та архітектури даних. Але він погано показує, хто саме користується системою і які цілі переслідує. DFD — це "погляд інженера", а не "погляд користувача".

> Щоб описати систему очима людини, нам потрібен інший інструмент — **Use Case (Варіанти використання)**. Саме цим ми займемося на наступній лекції.