仮説を型に

生成AI時代の曖昧さを後段へ流さない設計

株式会社アクティヴァーチ・コンサルティング
天野　悠

お約束

・本プレゼンは個人の見解であり、所属組織の公式見解ではありません。
・本発表は所属組織に寄与する意図で記載しておりません。

課題：型は後段へ流す前のスキーマ

データパイプライン

入力 → **検証** → 後段処理

不整合が後段で爆発

スキーマ不整合が通ると、
処理待ちやリトライ・例外調査が後ろで増える。

開発パイプライン

要求 → **型** → 実装

曖昧さが実装で増殖

要求が曖昧なまま流れると、
生成AIは未確定仕様まで高速に増やす。

顧客確認は必要だが、PoCは待ってくれない。
そこで、**設計時点で「許す形／許さない形」を型で先に決める。**

対策：Anyを使わない

悪い例：仕様を後回しにした形

```
def apply_discount(
    price: int,
    discount: Any,
) -> int:
    ...
```

良い例：業務の区別を形にする

```
from dataclasses import dataclass

@dataclass(frozen=True)
class FixedAmountDiscount:
    amount: int

@dataclass(frozen=True)
class RateDiscount:
    rate: float
    upper_limit: int

Discount = FixedAmountDiscount | RateDiscount
```

Anyが隠すもの

何でも受け取れるが、
何が来るか決められていない

やること

業務上の区別をコード上の区別にする。
たとえば、固定額と率の割引は別物。

先に決めること

種類	固定額か率か、それとも上限付き率か
禁止	同時指定・未指定を許すのか
追加	新種追加時にどこで検知するか

型は、決めたことしか守れない。

結論：曖昧さは、型と境界で前倒しして潰す

型は合意の代替ではなく、合意までの仮説として検証可能な形にする。

01 仮定を置く

仕様未確定でも、
種類・単位・範囲を仮決め。

仮説は後から変えてよい。

02 型に刻む

業務上の区別を
Union・dataclass・列挙型に。

無効な状態を表現させない。

03 境界で止める

外部入力・生成AI出力を
検証してから後段へ流す。

外れたら入口で止める。

現在の仮説を型に固定し、外れた値を入口で止める。
顧客確認で仮説が外れたら型を更新し、曖昧なまま実装を増やさない。