アーキテクト1年生が
アーキテクチャのことを語ってみる

> SW_Arch ソフトウェアアーキテクチャ集会LT 4/6

ありあな

自己紹介

ありあな(Alliana)

- 普段は TypeScript をメインに書いている
- ソフトウェアアーキテクチャを
  考え始めてちょうど1年ぐらい
- オレオレトレンド: 関数型プログラミング，DDD
- VRC 歴は1ヶ月半ぐらい (2026/02/21-)

GitHub: Allianaab2m

X: @Alliana_VRC

intro: この集会を始めたきっかけ

ソフトウェアアーキテクチャの
プラクティスが知りたい！

アーキテクチャなんもわからん

→ みんなどうしてるんやろ？

→ せや！集会開いてつよつよエンジニアから聞いたろ！

(エンジニア集会内で行われた主催同士の実際の会話)

今日話すこと

1. 「編集しにくいコード」の正体を知る
2. 立ち向かう武器 = アーキテクチャ（と、その注意点）
3. 明日からできる第一歩を試してみる

ちょっとメタい話

- (主催としての観点) 1回目の集会から具体的な話をするのは違う気がした
  - 初回に具体的な話をするとこの集会が
    「クリーンアーキテクチャ集会」や「DDD勉強会」になってしまう
  - ある程度の認識を合わせたいので，初回は抽象的なことを話したい
    基調講演的な感じ
- 私もそこまで理解しているわけではないです
  - アーキテクト1年生の振り返りトークだと思ってほしい！

「動いてるからいい」？

- 半年前のコード，**「なぜこう書いたか」**思い出せますか？
- 1箇所直したら，**想定外の場所が壊れた！**

ソフトウェア開発は1人ではできない

コードには必ず読み手がいる

- チームメンバー: あなたの暗黙知を持っていない人
- 3ヶ月後の自分: コンテキストを忘れた自分
- AI (Claude, Copilot): 意図が明確なコードは提案の精度を上げる

→ 複雑なコードは **時間** と **コンテキストウィンドウ** をどんどん奪っていく

コードの複雑さとは

複雑さ != コードが長いこと

- 1つ変更するために，**同時に** 多くのことを把握しないといけない状態
- 人間のワーキングメモリは大体 4 つ前後
    - 密結合な状態は脳のスペックを超えてしまう

複雑さを軽減するには？

→ コードを書くときに何かしらの仕組み，ルールが必要

アーキテクチャとは

一度に考える範囲を限定するための仕組み

- 関心ごとを分離 → **「ここだけ見れば OK 」** という状態を作る
- どのアーキテクチャにも共通して言えるのは **「関心の分離」**

手段の目的化には注意 ⚠️

アーキテクチャは目的ではない ← 重要！！！！

- 過剰な設計は新たな負債になりうる 本末転倒
- アーキテクチャを学ぶ意味 = **手札を増やすこと**
  - 知らないうちは **検討もできない**
  - 知った上で「今は必要ない」と判断できるのも設計力

試しにやってみる

try: 全部入りのハンドラ

```
async function handleSubmit() {
  // バリデーション
  if (!name || name.length < 2) { setError("名前は2文字以上"); return; }
  if (!email.includes("@")) { setError("メール形式が不正"); return; }

  // API呼び出し
  const res = await fetch("/api/users", {
    method: "POST",
    body: JSON.stringify({ name, email }),
  });

  // UI更新
  if (res.ok) {
    setUsers([...users, { name, email }]);
    setName("");
    setEmail("");
    setError("");
    showToast("登録しました！");
  } else {
    setError("保存に失敗しました");
  }
}
```

try: 関心を分離してみる

```
// バリデーションだけ
function validateUser(input: { name: string; email: string }): string | null {
  if (!input.name || input.name.length < 2) return "名前は2文字以上";
  if (!input.email.includes("@")) return "メール形式が不正";
  return null;
}

// API通信だけ
async function saveUser(user: { name: string; email: string }) {
  const res = await fetch("/api/users", {
    method: "POST",
    body: JSON.stringify(user),
  });
  if (!res.ok) throw new Error("APIコールに失敗しました");
}

// ハンドラは「組み合わせる」だけ
async function handleSubmit() {
  const error = validateUser({ name, email });
  if (error) { setError(error); return; }
  await saveUser({ name, email })
    .then(() => onSuccess())
    .catch(() => setError("保存に失敗しました"));
}
```

まとめ

1. 複雑さの正体 = **密結合**
2. 読み手 = **チーム，未来の自分，AI**
3. アーキテクチャは手段 知ることで **「意図的な選択」** の幅が広がる
4. 明日からできること 関心の分離を意識してやってみる

最後に

最適な設計は時と場合，プロジェクトによって異なりますが

意図のある選択はどんな状況においても良い結果をもたらします