# AI 時代のアーキテクチャの価値

somnicat

April 20, 2026

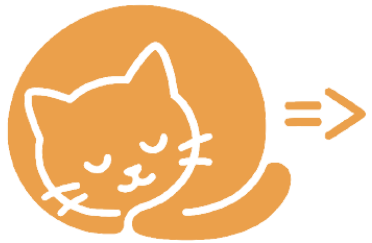
# 自己紹介

名前: **somnicat** (@somnicattus)

**職業**: Web アプリケーションエンジニア

**使う技術**: フルスタック TypeScript (Vue/Node.js)

最近 VRChat で技術系イベントの共同主催を始めた



# 今日話すこと

1. AI 生成コードへの懸念
2. AI が変えたもの
3. AI 時代のアーキテクチャ
4. 議論したいこと

# AI 生成コードへの懸念

# AI 生成コードへの懸念

LLM の台頭以降、AI が書くコードの品質についての議論が活発に

- **技術的負債**: 「ほぼ正しい」生成コードから生じる不確実性の蓄積
- **認知的負債**: 理解を伴わないコードが急増し、保守が困難になる
- **生産性の逆説**: AI 導入で逆に開発速度が低下するケースも報告

これらは本当に **新しい問題** なのか？

# 品質管理の課題は今に始まったことではない

- AI は **2つの文脈** で外注の代替として使われている:
  - 「より安価な労働力」—— 自分でもできるが、速く・安くやらせたい
  - 「自分より詳しい専門家」—— 自分にはない知識を借りたい
- 外注と AI に共通する既知の課題
  - 「ほぼ正しい」成果物
  - 理解の断絶
  - レビュー負荷

AI が本当に変えたものは

# 「理論の担い手」の消失

# Programming as Theory Building

Peter Naur が 1985 年の記事で主張したこと

> **ソフトウェアの本質はコードではなく、**
>
> **開発者の頭の中にある「問題解決のための理論」**

- 「なぜそうなっているか」というコードの背後にある判断と意図の体系
- 仕様書も実装も方法論も本質ではない

# AI は理論を担えない (今のところ)

- 今の AI は後から設計判断の根拠を問われても正確に回答できない
  - 「思い出す」わけではなく「現状から推測して再生成」になる
  - 「問題解決のための理論」の構築には膨大なコンテキストが必要
- 今の AI は運用の柔軟性という面で人間の頭に勝てない
  - コンテキストウィンドウの限界
  - セッションは揮発する
  - いちいちファインチューニングできない

# 外注と AI の構造的な違い

| | 人間への外注 | AI への委譲 |
|---|---|---|
| 理論の所在 | 担当者か外注先に存在する | **どこにも存在しない** |
| 判断の問い合わせ | 外注先に質問できる | **生成された説明のみ** |
| 暗黙の設計判断 | 対話で引き出せる | **出力に現れない** |
| 知識の蓄積 | チームに経験として残る | **セッションごとにリセット** |

- 人間への外注では、理論は少なくとも外注先に存在した。
- **AI への委譲では、理論を持つ主体がどこにも存在しない**

# 個人の生産力 × 理論不在のジレンマ

- AI は **個人の生産力** を桁違いに拡大した
- だが理論を構築する **認知的コスト** は変わらない
- 一人が AI で 100 人分のコードを生成できるようになったとしても、100 人分の理論を一人で担えるのか？
- 理論の構築を諦めたとき、AI が生成したソフトウェアは本当に「資産」といえるのか？

# 改めて —— あの「負債」の正体

- **技術的負債**: 従来の技術的負債は「意図的な近道」
  - AI による負債は **理論を持つ者がいない負債。返済の見積もりなどない**
    - **GIST 負債** (GenAI-Induced Self-Admitted Technical Debt)
- **認知的負債**: 従来の理解の断絶はコミュニケーションで解消できた
  - AI 委譲では **問い合わせ先が存在しない。再構築できる理論もない**
- **生産性の逆説**: 品質管理コストは昔からあった
  - AI が生成する理論のないコードの品質管理は困難
    - **品質管理コストの増加が生産コストの削減を上回る場合も**

# AI 時代にアーキテクチャに求められること

アーキテクチャは「理論の器」の問題を解決できるか

# アーキテクチャは「理論の器」に"なれない"

- Naur の言う「理論」は本質的に **人間の頭の中** にしか存在できない
  - アーキテクチャという「方法論」が理論を代替することはできない
- 判断の文脈は形式化しきれない
  - なぜこの境界か
  - なぜこの抽象か
- 仕様書・テスト・型定義に書き下せるのは **理論の断片** にすぎない

# それでもアーキテクチャにできること

## 1. 理論の骨格を担う

- 型・テスト・制約定義は理論の全体ではないが、**骨格** として機能する
- 「AI ネイティブ開発では **テストライブラリこそが資産**」

## 2. 人間の理論構築を助ける

- 認知負荷が下がれば、人間が理論を構築する余裕が生まれる
- アーキテクチャは理論の器ではなく、理論を **育てる足場**

# 自動化された理論の検証装置

テストが書きやすい設計で **理論の検証を自動化する**

- **ヘキサゴナルアーキテクチャ (Ports & Adapters)**
  - 外部依存を境界の外に追い出し、ビジネスロジックを純粋に保つ
- **関数型アーキテクチャ**
  - 副作用のない純粋関数にロジックを集約する
  - 入力と出力だけでテストでき、テストを **実行可能な仕様** にできる
- **依存性の注入 (DI)**
  - 結合点を明示し、テスト時に差し替え可能にする

# 境界が認知負荷を下げる

DDD の **Bounded Context** = 「ここから先は別の世界」という宣言

- ドメインの境界は AI の暴走を防ぐ壁
- 人間が理論を構築できる自然な単位を作る
- 境界の中に閉じていればレビュー範囲が限定される
  - 境界の内側: AI に任せやすい
  - 境界の定義や横断: 人間の理論が必要

# 議論

# 議論したいこと

1. AI に委譲した仕事の「理論」を、誰がどう保持していますか？
2. 人間への外注と AI への委譲で、管理方法を変えていますか？
3. ここまでは AI に任せる、という線引きをどこに置いていますか？
4. 理論を育てる「足場」として、何が最も効果的だと感じていますか？

# Thank You

# References


- Developers remain willing but reluctant to use AI: The 2025 Developer Survey results are here
  - https://stackoverflow.blog/2025/12/29/developers-remain-willing-but-reluctant-to-use-ai-the-2025-developer-survey-results-are-here/
- “TODO: Fix the Mess Gemini Created”: Towards Understanding GenAI-Induced Self-Admitted Technical Debt
  - https://arxiv.org/html/2601.07786v1
- Programming as theory building
  - https://www.sciencedirect.com/science/article/abs/pii/016560748590032 8
- AI-Native Engineering Manifesto
  - https://github.com/Ge-limin/ai-native-engineering-manifesto