

LIGHTNING TALK

# AI活用のコスパを最大化する方法

トークン制約時代の依頼設計・CLI運用・共有資産化

2026 / Engineering Productivity

## 今日のゴール

**01**

再試行を減らす依頼設計

**02**

GUI依存からCLI運用へ

**03**

個人知見を共有資産にする

キーワード: 品質の再現性 完了までの総コスト チーム展開

## なぜ今「コスパ設計」なのか

生成AIの本質は **質問回数** ではなく **再試行削減**  
トークン/リクエスト制限下では、1回の精度が生産性を決める  
曖昧な依頼は手戻りを生み、総コストを押し上げる



やり取り回数



認知負荷



完了速度

## コスパは「単価」より「総コスト」で見る

### 見落としがちなコスト

要件整理の時間  
再修正の往復回数  
レビュー差し戻し  
セキュリティ確認工数

### 判断基準

**完了までの時間 × 往復回数**

同単価でも、完了が半分なら実質コスパは大幅改善

## まず避けるべき3つの失敗

### 要件が1文

「いい感じに直して」では優先順位が不明

### 制約未定義

文字数・読者・形式がないと追加往復が増える

### 一括で巨大依頼

調査～検証を同時要求すると精度が落ちる

失敗の共通点: 「伝える」だけで「伝わる設計」になっていない

# 伝わる依頼文の最小テンプレート

## # 目的

- 何を達成したいか

## # 前提

- 現在の状況
- 使用環境

## # 制約

- 使ってよい技術／禁止事項
- 納期・優先順位

## # 期待する出力

- 形式（Markdown / JSON / コード）
- 完了条件

構造化すると、推論ミスと認識ズレを減らせる

## 複雑依頼は4フェーズ分割

### 1. 調査

情報整理・用語定義・前提確認

### 2. 設計

構成案・採用理由・非採用理由

### 3. 実装

コード/本文の生成

### 4. レビュー

抜け漏れ・要件充足・品質統一

品質が落ちた地点を切り分けやすく、結果的に最短で終わる

# GUIよりCLIを主軸にする理由

## CLIの優位性

- パフォーマンスが良いケースが多い
- 成果品質が安定しやすい
- トークン効率が良い場面がある
- 裏で実行でき、マルチタスクしやすい

## 運用面の価値

- 実行履歴を残せる
- 手順の再現性が上がる
- チーム共有がしやすい

# チームで効く「AI共有資産」

依頼テンプレート

検証手順

レビュー観点

カスタムプロンプト

スキル定義

## 共通化する部分

目的・前提・制約・完了条件

## 製品依存で調整する部分

ツール呼び出し

出力制約

コンテキスト管理

## セキュリティ前提の運用ルール

個人情報・顧客情報・機密情報は入力しない  
必要時は必ずマスキングして投入する  
ログ保存範囲と公開範囲を事前確認する  
社内ガイドラインに沿った運用フローを明文化する

無料版活用のメリットはあるが、情報統制を最優先する

## まず着手する最小セット

**1**

依頼テンプレートを1枚作る

**2**

複雑依頼を4フェーズに分割

**3**

CLIで実行履歴を残す

### 効果測定KPI

完了までのやり取り回数

1タスクあたり所要時間

レビュー差し戻し件数

# まとめ

AI活用の成果は、ツール単体より **運用設計** で決まる

3本柱: **依頼設計** **CLI運用** **共有資産化**

最初は小さく始めて、測って改善する

**再現可能な実務プロセス** に変えることが、最も大きな投資対効果。

参考: [OpenAI Codex CLI](#) / [GitHub CLI](#) / [Qiita関連記事](#)