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年2月

📋 Today's Agenda

🎯

柱① 依頼設計

伝えるではなく、伝わる文章を設計する

🤝

柱③ 共有資産化

個人最適をチーム最適に変える

💻

柱② CLI運用

高パフォーマンス運用を設計する

✅

まとめ — 最小セット3つ

今日から始められること

🔍 なぜ「コスパ設計」が生産性を左右するのか

⚠️ **生成AIは 「無限に使える道具」ではない ― トークン制限・リクエスト制限がある**

❌

曖昧な依頼

追加指示が増えトークン消費が拡大する

⚠️

品質のばらつき

依頼者ごとに結果が変わりチーム再利用困難

✅

設計された依頼

1回あたりの成功率↑ 認知負荷↓

💡 **AI活用の本質 ＝ 「たくさん質問する」ではなく「再試行を減らす設計」**

💸 コスパ＝「単価」でなく「完了までの総コスト」

❌ 誤った見方

1リクエストあたりの料金**だけ**を見る

```
<h2 style="margin:.875rem 0 .75rem;">✅ 正しい総コスト</h2>
<div class="card card-green">
  <ul style="margin:0;font-size:.88rem;padding-left:1.1rem;">
    <li>要件整理にかかった <strong>時間</strong></li>
    <li>出力の <strong>再修正回数</strong></li>
    <li>チーム内レビューの <strong>手戻り</strong></li>
    <li>セキュリティ確認の <strong>工数</strong></li>
  </ul>
</div>
```

1/2

完了時間が半分になれば
実質コスパは **2倍以上** 改善

安くても往復が増えれば
トータルで高くつく

柱① 依頼設計

> 伝えるではなく、「伝わる文章」を設計する

📝 最低限の依頼テンプレート

```
# 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
```

🎯 目的 + 前提

AIが「何のために」を理解するための軸になる

🚧 制約

禁止事項・優先順位を明示して推論ミスを防ぐ

📤 期待する出力

形式と完了条件を決めると品質が安定する

> 💡 **自由記述より構造化された要件のほうが推論ミスが少ない**

✂️ 複雑な依頼は「4フェーズ分割」

1. **🔍 調査フェーズ**
   既存情報の整理・用語定義・前提確認
2. **📐 設計フェーズ**
   構成案・採用方針・非採用理由の整理
3. **⚙️ 実装フェーズ**
   コードまたは本文の生成
4. **🔎 レビューフェーズ**
   バグ・抜け漏れ・文体統一・要件充足の確認

> 🚀 1回で完璧に出すより分割して精度を上げるほうが、最終的に速く終わる

🌐 日本語 vs 英語の使い分け Tips

英語が向くケース

- 英語の技術情報が豊富な領域
- 出力の安定性を優先するとき

日本語出力が必要なら明示する

```
- 調査・推論は英語情報を参照
- 最終出力は日本語で作成
- 専門用語には補足を入れる
```

✅ 依頼品質チェックリスト

- ✓「誰向けか」を書いたか
- ✓「何をもって完了か」を書いたか
- ✓出力形式を明示したか
- ✓禁止事項を入れたか

柱② CLI運用

GUIよりCLIを選ぶべき理由

💻 CLIを選ぶ4つの理由

⚡

パフォーマンス

GUI比で応答が速い場合が多い

🎯

成果の品質

コンテキスト制御が精密で出力が安定

💰

トークン効率

余分なUI往復なしでトークンを節約

🙌

マルチタスク化

自分の作業の裏でAIを並行実行できる

📌 **GitHub Copilot CLI・Codex CLI などがVS Code拡張より優位な場面が多い**

柱③ 共有資産化

> 個人最適をチーム最適に変える

🤝 チームのAI共有資産設計

共有すべき資産の種類

- 📄 依頼テンプレート（Markdown）
- ✅ レビュー観点チェックリスト
- 🔁 よく使う検証手順
- 💬 カスタムプロンプト
- 🤖 スキル定義・カスタムエージェント設定

> 🎓 **新人でも一定品質に到達しやすくなる**

エージェント差分への対応

共通部分（先に標準化）	製品依存部分
目的・前提・制約・完了条件	ツール呼び出し仕様
どのAIにも通用するコア	出力制約・コンテキスト管理

製品変更時は **差分だけ** 調整すればよい
Claude / Codex / GitHub Copilot…

🔒 セキュリティ前提の運用＆形骸化対策

必須ルール

機密情報は入力しない
個人情報・顧客情報・機密情報はNG

必要ならマスキング
匿名化・ダミー化してから投入する

運用フローを明文化
社内ガイドラインに沿った手順書を整備

形骸化しないコツ

ルールが多すぎて使われない →
最小セット（テンプレ1枚＋観点3項目）から始める

更新されず古くなる →
月1回・15分のメンテ時間を固定する

ブラックボックス化 →
変更履歴と意図を短く記録する

✅ まとめ：今日から始める最小セット

1. 📄 **依頼テンプレートを1枚作る**
   目的・前提・制約・期待出力の4項目を固定するだけでOK

2. 🎀 **複雑依頼を4フェーズに分割する**
   調査 → 設計 → 実装 → レビュー の順で進める

3. 💻 **CLIで実行履歴を残す**
   手順の再現性を確保し、チーム共有しやすくする

📊 **効果測定の指標：** ① やり取り回数　② 1タスクあたりの時間　③ レビュー差し戻し件数

🙏 ご清聴ありがとうございました

依頼設計 × CLI運用 × 共有資産化

この3本柱でAI活用を

「属人芸」 から 「再現可能な実務プロセス」へ

📄 元記事 → Qiita「AI活用のコスパを最大化する方法」

参考: GitHub CLI / OpenAI Codex CLI 公式ドキュメント