Agentic Coding Recommendations

鄭道鉉（Roboco）· 2025-06-26

アジェンダ

1. Why Agentic Coding?
2. The Basics
3. Language Choice
4. Tools, Tools, Tools
5. Speed & Stability
6. Simplicity First
7. Parallelization
8. Refactoring Mindset
9. Key Takeaways

The Basics

- **権限確認を無効化** → 高速ループ
- **MCPは最小限**
  - 通常のシェル/スクリプトで十分なら、そのまま使用
- **Dockerサンドボックス推奨**（ リスク軽減 ）
- **IDE依存を最小化** → Vim + 最終編集

Language Choice

> "Go is sloppy — perfect for agents." – Armin Ronacher

- **Go推奨**
  - `context` パターンが明示的
  - テストキャッシュで高速ループ
  - 構造的インターフェース → LLMフレンドリー
  - エコシステムの変化が少ない
- Python/Rustは **解釈·ビルド遅延**とマジックで難易度↑

Tools, Tools, Tools

- **何でもツール（ スクリプト·ログ·MCPサーバー ）**
- 核心4原則
    i. 高速であること
    ii. 明確なエラーメッセージ
    iii. 失敗 (クラッシュ) はOK、ハングはNG
    iv. 観測性·デバッグ容易性
- `Makefile` 活用:
    - `make dev` 、`make tail-log` 例
    - **PIDロック**で重複起動防止

Speed Matters

- 推論コスト + ツール遅延 = 生産性低下
- **コンパイル·ブート時間最小化**
  - 軽量スクリプト → 3 ms
  - 重いサービス再起動 → 60 s ❌
- 必要時は**ホットリロード用デーモン**作成
- ログは**簡潔 + 有意義 + トグル可能**

Stability & Copy/Paste

- **依存関係のアップグレード**は保守的に
  - LLM breadcrumb / コメントとの衝突防止
- "Build It Yourself" – ライブラリより**直接実装**を優先
- 固定されたエコシステム = 予測可能なコード生成

Write Simple Code

- "Dumbest possible thing that works"
- 長い名前の**関数中心**コード
- 継承を避け、**ORMよりSQL**
- **権限チェック**はコード近くに！（ AIが認識しやすく ）

Make It Parallelizable

- エージェント単体速度は遅い → 多重インスタンス
- 状態分離：別チェックアウト·DB·Redisシャード
- `container-use` 等MCPでコンテナ分離実験

Learn to Refactor

- タイミングが肝心
  - 早すぎると無駄、遅すぎるとLLM混乱
- 複雑度↑ → コンポーネントライブラリ分離タイミング
- リファクタ段階でもエージェント積極活用

Key Takeaways

1. **速度**：短いフィードバックループ
2. **シンプル性**：Go、シンプルなツール、明確なコード
3. **観測性**：ログ + PID保護
4. **安全な並列化**：コンテナ分離
5. **適時リファクタ**で複雑度制御

References

- Armin Ronacher, "Agentic Coding Recommendations" (2025-06-12) https://lucumr.pocoo.org/2025/6/12/agentic-coding/
- その他：Playwright-MCP、container-use、Cursor Background Agents

Q&A

ありがとうございました！