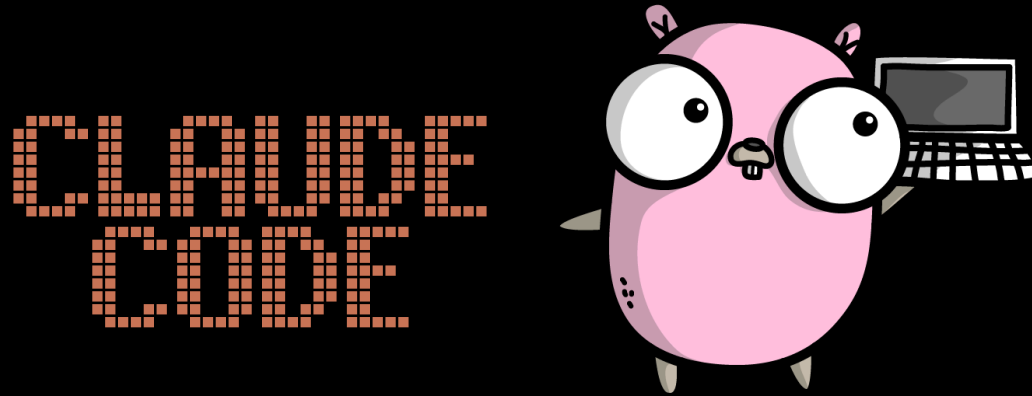


Accelerating Go Development with Claude Code



Wojciech Barczynski

Development with AI

- AI boosts productivity.
- Tools and models evolve rapidly.
- The challenge is to find what works.

Goal

- Share a pragmatic AI workflow for Go development.
- Discuss effective strategies and tools.



+ Tools

Models

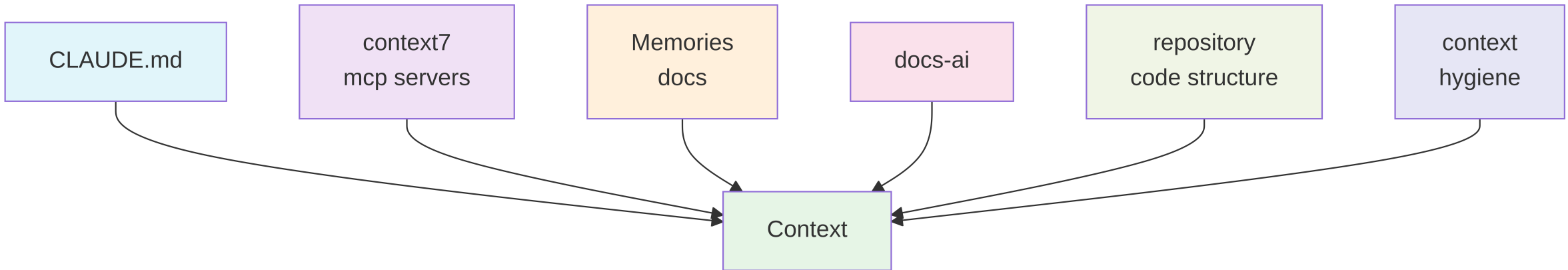
- Anthropic models lead
- `claude` → better results
- [Cut-off](#) - march 2025

Models

Models have strengths and weaknesses:

- Claude Code
- Gemini

Context



CLAUDE.md

- Keep it up-to-date.
- Update it when new features are added.
- Create a command for easy updates.

CLAUDE.md


For Go-specific context, include:

- Project coding conventions.
- Go design patterns, with examples (e.g., for error handling).

Plan.md

- Keep the model on track.
- When in doubt, create it.
- A must for anything more complicated.
- Benefits for the model.
- From scratch or based your input/initial plan.

Plan.md

1. Improve the Plan.md
2. Ask Claude to open a PR for step 1
3. Ask Claude to do the implementation
4. Iterate with Claude on code/plan/...
5. Ask Claude to update docs/CLAUDE.md
6. Mark  completed tasks
7. Squash & Merge
8. Clear context
9. Repeat

context7 mcp

- Fetches on-demand documentation and code snippets.
- Additionally:
 - Add links to [prompts](#).
 - Add information to `memory/`.
 - Or save to `docs-ai/`.

.claude/memory

Memory (`.claude/memory`):

- Convention, not automatically read ([docs](#)).
- Use for one-off prompts (e.g., `migration_sqlite_to_psql.md`).
- Store best practices.
- Save prompts for future use (e.g., `memory-template`).

docs-ai / ai-docs

- More extensive docs and larger mds.
- You can link them in `CLAUDE.md`.

Code structure / Repository

1. Vertical code structure 🌞
2. Modular design
3. Layered architecture / numerous dependencies 🦴

`CLAUDE.md` in subfolders.

Context

```
Read .claude/memory/* and ... use command ...
```

context hygiene

Once the task is complete or a session is too-long:

- Save any essential information
- Clear the context using the `/clear` command
- `git worktree` for isolated environments
- `/context`

subagents

```
---  
name: your-sub-agent-name  
description: Description of when this subagent should be invoked  
tools: tool1, tool2, tool3 # Optional - inherits all tools if omitted  
model: sonnet # Optional - specify model alias or 'inherit'  
---
```

Your subagent's system prompt goes here.

subagents

Examples:

- Frontend
- Code Reviewer
- Project Mgmt
- Various tools to manage them, e.g., [ccagents](#)

Prompt for Claude Code

- The CO-STAR and CLEAR Framework
- Keywords, e.g., exactly, detailed, [...](#)
- Role-task format pattern

```
You are a [ROLE] with expertise in [DOMAIN].  
Your task is to [SPECIFIC_ACTION].
```

The CO-STAR Framework

- **C**ontext: Background information
- **O**bjective: The purpose or goal
- **S**tyl: Formal, informal, etc.
- **T**one: Friendly, authoritative
- **A**udience: for whom it is
- **R**esponse Format: output

The CLEAR Framework

- **C**oncise: Be brief and to the point.
- **L**ogical: Structure your prompt in a logical order.
- **E**xplicit: State exactly what you want.
- **A**daptive: Frame the prompt to be adaptable to different scenarios.
- **R**ole-based: Assign a role to the AI.

1. Task context

2. Tone context

3. Background data, documents, and images

4. Detailed task description & rules

5. Examples

6. Conversation history

7. Immediate task description or request

8. Thinking step by step / take a deep breath

9. Output formatting

10. Detailed response (if any)

Will help:

- Good to watch 1-2 videos about prompt engineering
- [prompt optimizer](#) at [claude.ai](#)
- Claude can review your prompts as well.

Claude Code

- **ESC**: Provide additional information.
- **ESC ESC**: Cancel the current action.
- **Planning Mode**: Deconstruct complex tasks into smaller steps.

Code as part of the prompt

- Write your test first (~TDD)
- Define interfaces between components

Claude Code Tools

- **Hooks:** Customize behavior with pre/post-action scripts.
- **OpenTelemetry:** Integrated for observability and performance monitoring.
- **ccusage**: CLI tool to track token usage and costs.

Choosing Your Tools

How I approach it:

1. **CLI Tools** (`gh`, `rg`, `eza`, ...): Fast and efficient for common tasks.
2. **Python Scripts** (with `uv`): Best for automation and complex logic.
3. **Mcp** few use cases.

Go-Specific

Claude benefits from Go's rapid feedback loop:

- **Strongly-typed language:** Catches errors before runtime.
- **Strict formatting:** Enforced by tools like `gofmt` and `golanci-lint`.

I typically use [Claude hooks](#) to automate these checks.

Accelerating Go Development

- A Continuous Process

Accelerating Go Development

- Share what you have learned with your team.
- At Unoperate, we run AI retrospectives weekly.

Accelerating Go Development

- The more verticals your app has, the easier it is for Claude.

Accelerating Go Development

- Model
- Context
- Prompt
- Tools

Demo → Claude

github.com/wojciech12/talks & wbarczynski.pl

Thank you
unoperate

Backup Slides

Prompt Engineering

- [Prompt Best Practices](#)
- [Prompt library](#)
- [CO-STAR- Article on prompting](#)