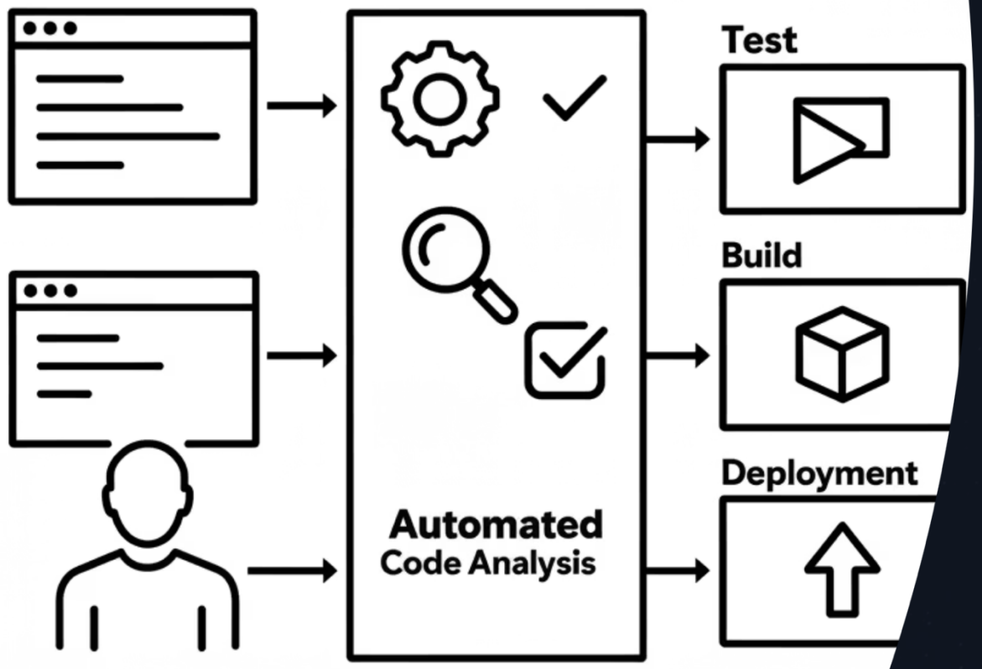
A Practical Workflow for AI Coding Assistants

Demonstrated through building 2D arcade games—real code, real workflows, real results

Jon Green — Stanford Engineering CGOE

Special Thanks

Ray Saray, Ali Karim, Rafael Cruz



Beyond Prompt + Copy-Paste

Codebase-Aware

Pulls relevant files, APIs, usage patterns.

Agent Loops

Runs tools, executes commands, iterates until checks pass.

Direct Changes

Opens PRs, multi-file refactors.

Modern Coding Assistants

Embedded Agents

IDE/CLI agents. Repo context. Propose/apply changes.

Leading Examples

Claude Code, Gemini, Copilot (CLI/Workspace), Cody, Cursor, Replit.

Project-wide

Multi-file refactors. Complex project changes.

Beyond autocomplete. Repo-wide context. Multi-file edits, autonomous PRs. Understand project structure & conventions.

Why AI Assistants Matter Now

55%

Faster Task Completion

GitHub Copilot controlled study: developers completed coding tasks 55% faster than control group.

2x

Coding Speed Boost

McKinsey pilots show developers can complete coding tasks up to twice as fast with generative AI.

46%

Trust the Accuracy

Stack Overflow 2024: Only 46% of developers trust AI tool accuracy, despite widespread adoption.

References:
- Microsoft Research: 'The Impact of AI on Developer Productivity: Evidence from GitHub Copilot'
- McKinsey: 'Unleashing developer productivity with generative AI' (2023)
- Stack Overflow Developer Surveys 2024-2025

Live Demo

Before the demo: Check the repo—no hidden tricks. The code you'll see is exactly what's here.

https://code.stanford.edu/jon.b.green/ai-arcade-demo

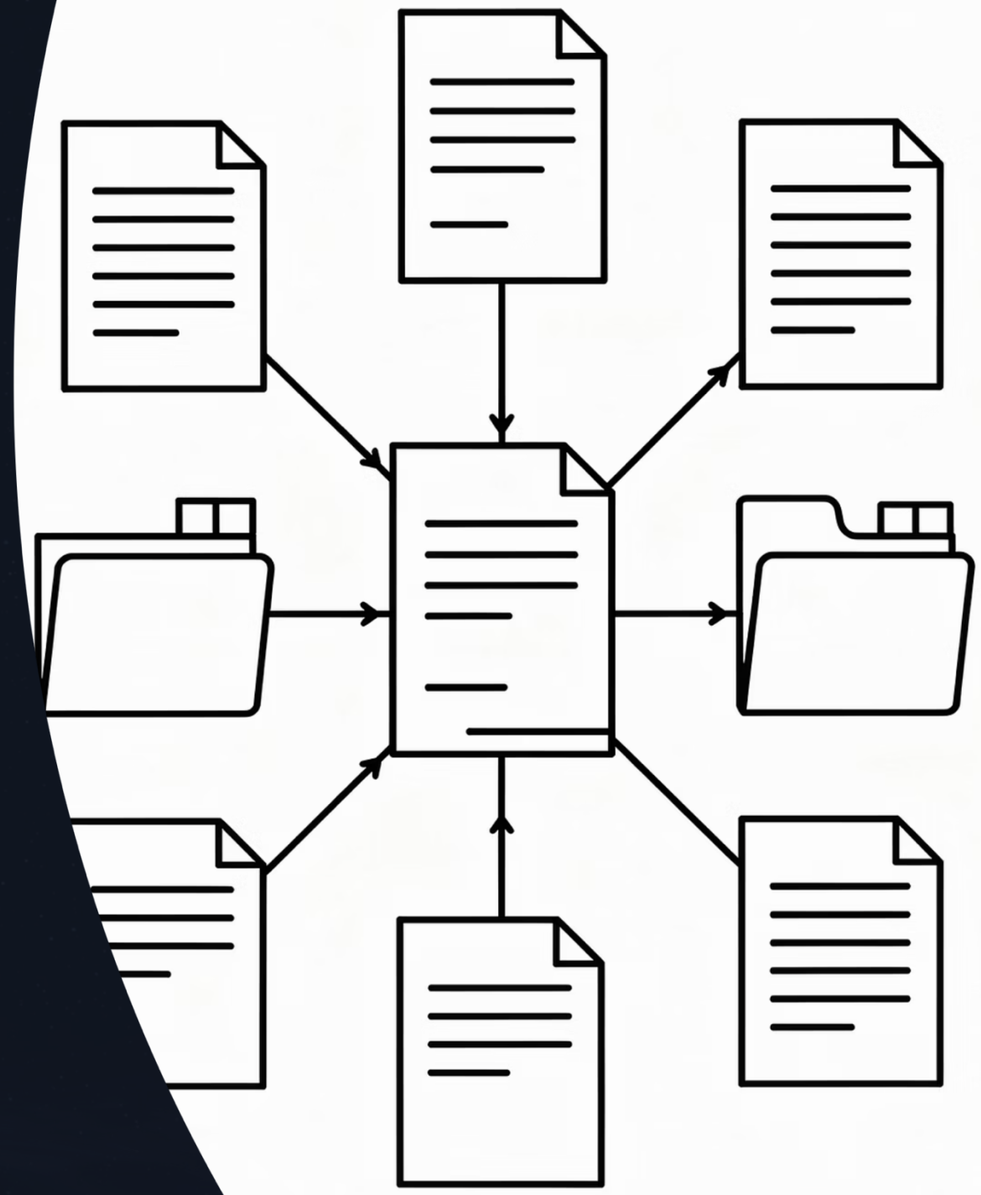
Revealing the Tricks

Spoiler: I'm not a magician

Repo-native workflow: Notes, TODOs, guides, tests. Self-reinforcing. Versioned in Git.

Notes

Context & Continuity



Session Notes: What & Where

Purpose

Instant continuity. Resumable in seconds.

Captures living context: decisions, trials, failures, next steps (beyond Git).

Location & Structure

Store in `.session-notes/` (repository root).

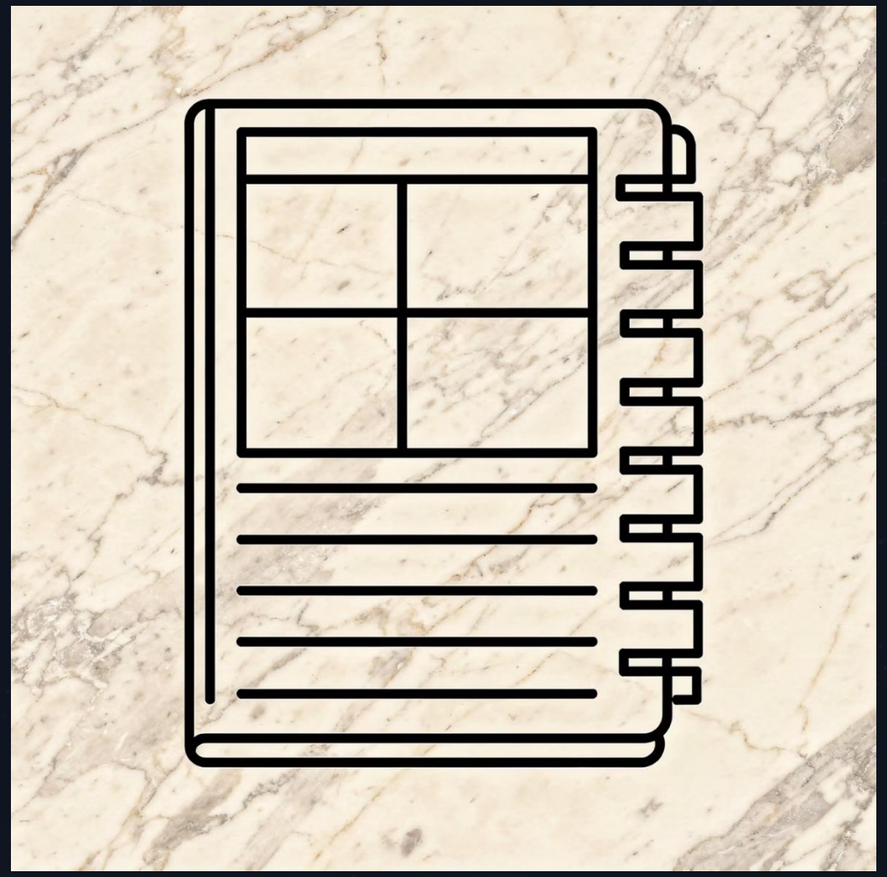
One file per day per work thread.

Naming Convention

`YYYY-MM-DD-task-description.md`

Consistent task names daily.

- `2024-01-15-api-auth-refactor.md`
- `2024-01-16-api-auth-refactor.md`
- `2024-01-17-api-auth-refactor.md`



Triggers & Habits for Session Notes

01

Before Handoff

Note before context switch.

02

After Commit

Log commit ID, changes.

03

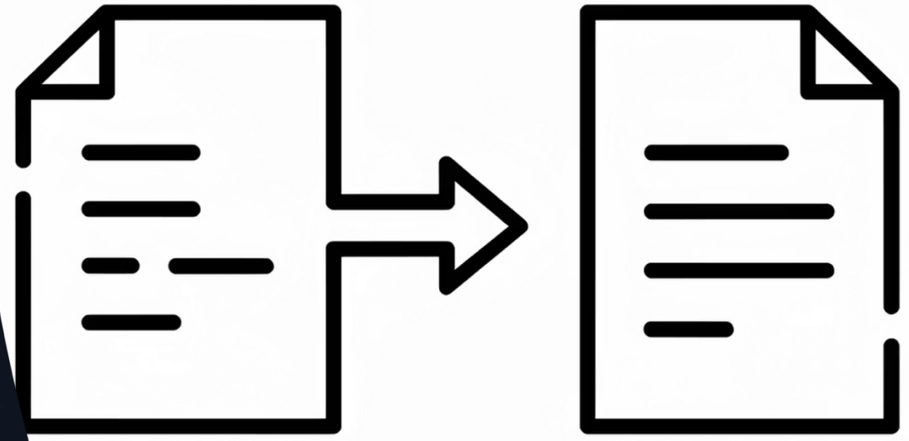
After Errors

Record errors, lessons.

04

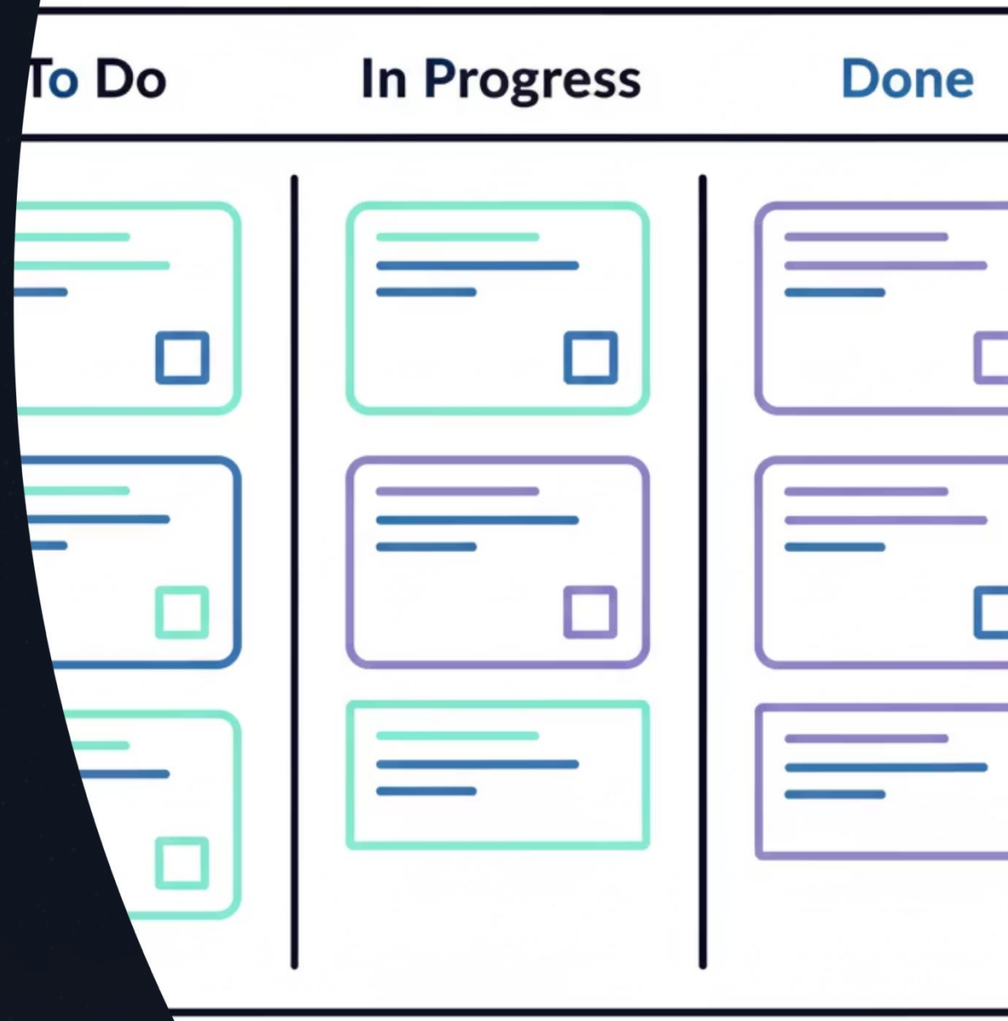
During Work

Timestamped bullets: progress, issues, changes.



TODO Tasks

Structured specs that AI assistants understand



TODO Tasks System Structure

Folders

- `todo/` folder
- `TODO-INDEX.md`: Master list, status, priority.
- Tasks: `{NNN}-{task-name}.md`

Naming

- Tasks: `001`, `002`, `003`…
- Recurring: `R001-name.md` (reusable)

Status & Priority

- Status: ❌ Not Started | ⏳ In Progress | ✅ Complete
- Priority: 🔴 High | 🟡 Medium | 🟢 Low

Numbering ensures order. Index: single source of truth for key tasks.

TODO Lifecycle & Archive Strategy

1. **Create**

   Template, number, index, priority.

2. **Review**

   Developer reviews AI-generated initial version, refines requirements.

3. **Build**

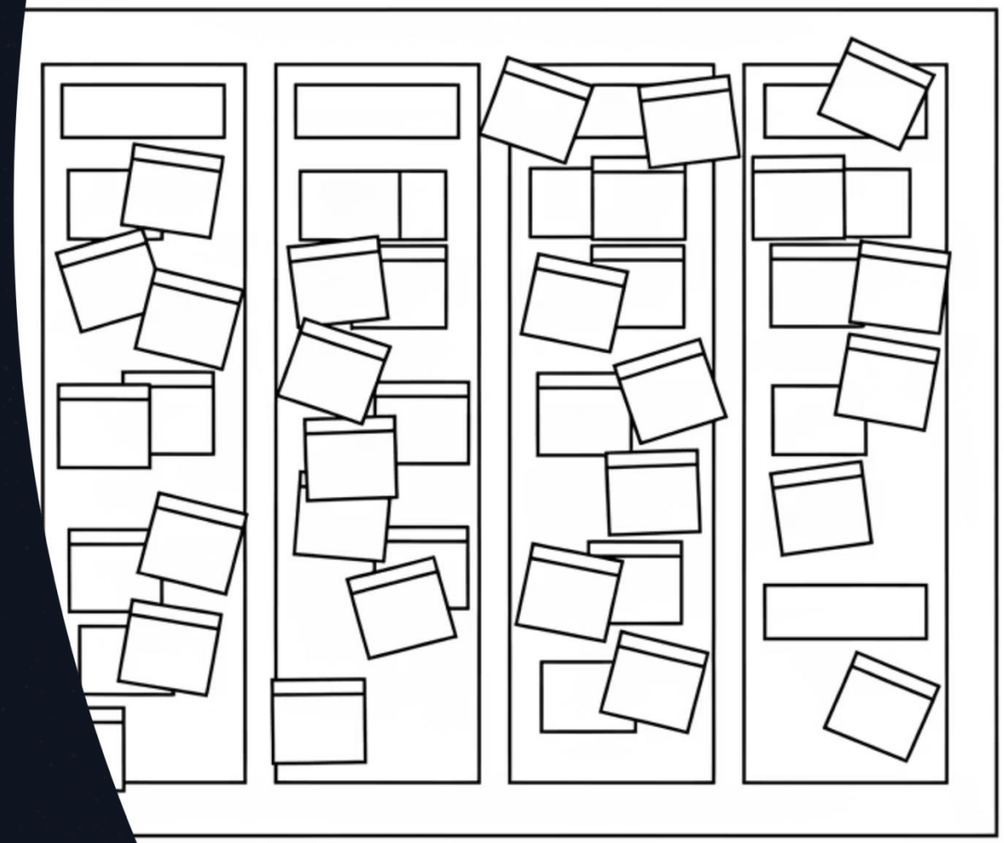
   Developer and AI collaborate iteratively to implement the feature together.

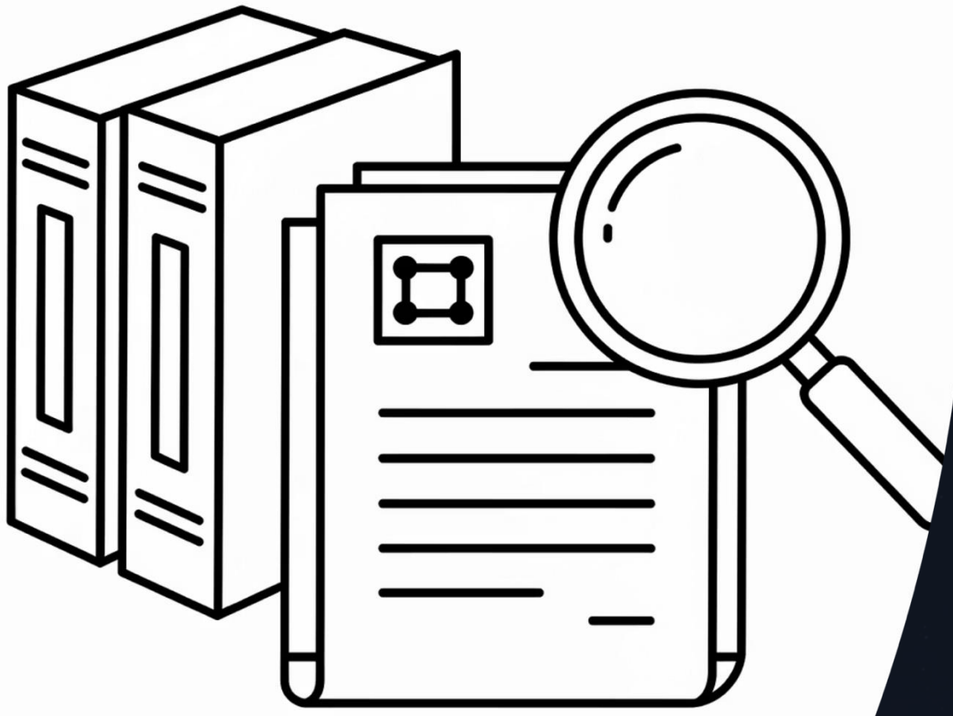
4. **Done**

   Deployed, tested, docs, confirmed.

5. **Archive**

   Commit hash, retrieval cmd, move to archive.





Guides

Documentation

Guides: Docs-as-Code Structure

📂 Structure & Naming

- `docs/guides/` (category subfolders):
  - `processes/` (workflows)
  - `troubleshooting/` (fixes)
  - `patches/` (emergency)
  - `tutorials/` (learning)

Files: `GUIDE-{topic}.md` (lowercase, hyphens)

🎯 Creation Process

Start from `TEMPLATE-guide.md`.

Update guides index README.

Guides **versioned with code**; evolve with system.

Guides capture solutions & knowledge. Stored in Git with code: ensures sync and review.

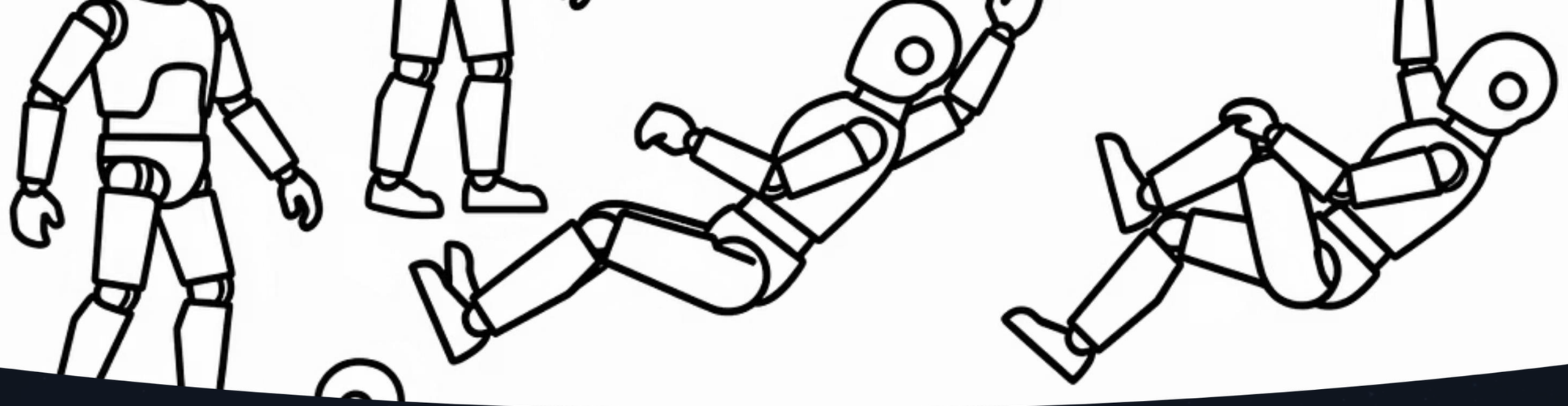
Guide: Security

Never Include Secrets.

Always Use Placeholders.

Automated Protection:

- Use automated scripts to scan for sensitive information before committing
- Run validation checks in pre-commit hooks to catch accidental leaks



Tests as Guardrails

Test Layers, Layout & CI Integration

Test Structure

- `tests/unit/`: Fast, isolated
- `tests/integration/`: Component interaction
- `tests/e2e/`: User journeys
- `tests/smoke/`: Health checks
- `tests/security/`: Vulnerability scans

Test Commands

- Runners: pytest, npm, dotnet
- Make targets: `test-unit`, `test-integration`
- Granular control

CI Gates

- PR: Unit + Integration + Smoke
- Nightly: E2E + Security scans
- Balance speed/thoroughness

What's Next

Tomorrow's Development Stack

What's Next?

1

Today: Manual Text Files

- Session notes, TODO tasks, guides stored as markdown files
- Requires manual orchestration

2

Tomorrow: Integrated Tooling

- AI agents automatically maintain session context and notes
- Reference guides without manual file management

3

Long-term: Autonomous Development

- Context repos and guides maintained by agent automatically
- Non-technical users define requirements in task systems
- Testing built into coding agent instructions

The Developer's Evolving Role

Review & Evaluation

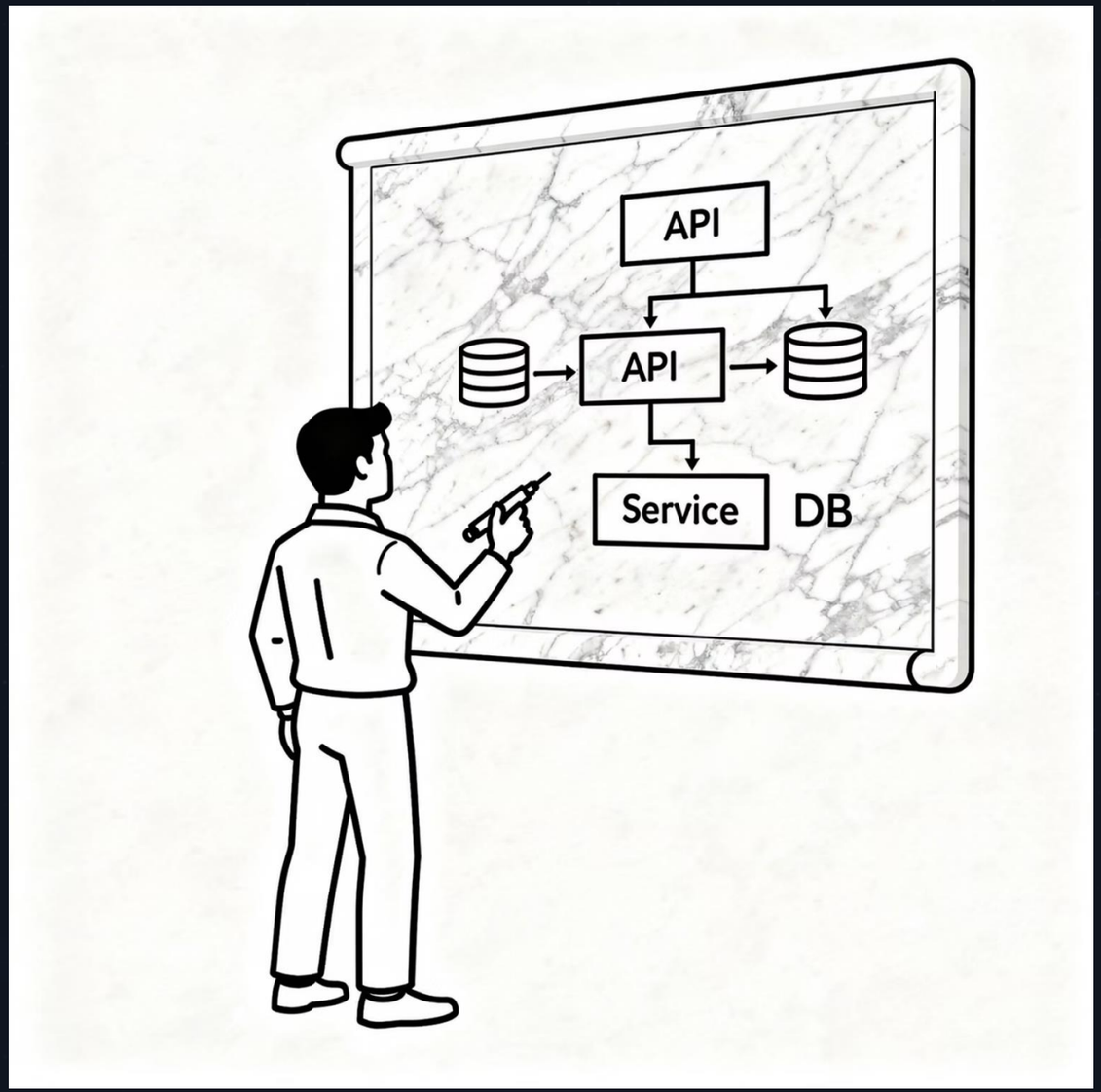
Assess AI-generated code for correctness, security, and maintainability.

Testing & Validation

Design test strategies to verify AI output meets requirements and handles edge cases.

System Design & Architecture

High-level structure, patterns, and constraints—the conceptual foundation.



Access the Workflow

Scan the QR code or visit the repository to explore the full CLI AI workflow, examples, and documentation.

https://code.stanford.edu/jon.b.green/ai-arcade-demo

Questions & Discussion

Let's Connect

Jon Green

Stanford Engineering CGOE

Email: jon.b.green@stanford.edu

LinkedIn: linkedin.com/in/jgreen01

Scan to connect on LinkedIn