DEVELOPER-FIRST · LEARNING-FIRST · HUMAN-OWNED

THE DEVELOPER-FIRST AI WORKFLOW

Version 2.0 Focused · Momentum Edition

A practical operating philosophy for learning, building, reviewing, and automating without surrendering engineering ownership.

NORTH STAR — **Use AI to increase my leverage as an engineer - never to decrease my ownership of the work.**

YOU · PRINCIPAL ENGINEER

Own judgment, architecture, implementation decisions, learning, and final review.

CHATGPT · PROJECT ROOM

Persistent context, files, instructions, and ongoing learning/building conversation.

TOOLS · EARNED LATER

Sonnet for narrow repeatable reviews. Codex for bounded execution with review gates.

THE PRACTICAL STACK

GPT-5.6 Sol Medium in ChatGPT Plus for daily serious work. Sonnet API for future utility tools. Codex later, when bounded execution becomes useful.

LEARN DEEPLY. BUILD INTENTIONALLY. AUTOMATE DELIBERATELY.

01 · WHAT THIS VERSION CLARIFIES

The original Version 2.0 was broad and model-agnostic. This focused edition keeps the same depth and philosophy, but centers the actual stack I expect to use for now: ChatGPT Plus with GPT-5.6 Sol Medium, Sonnet API as a future utility layer, and Codex as a later executor.

> **THE MAIN CORRECTION** — **The workflow does not need a pile of model tiers to be good. It works because it is project-first, learning-first, and human-owned.**

The stack is intentionally small

Layer	Role	Why it exists
You	Principal engineer	Own judgment, architecture, implementation decisions, learning, and final review.
ChatGPT Plus	Collaborative project workspace	Hold project context, files, instructions, and the ongoing learning/building conversation.
GPT-5.6 Sol Medium	Serious daily driver	Handle most coding, debugging, explanation, project planning, test design, and learning conversations.
Sonnet API	Future utility layer	Power narrow tools that package selected project context into repeatable reviews and reports.
Codex - later	Bounded executor	Eventually implement clearly scoped repository tasks on branches with tests and review gates.

The invariant

Models and product names may change. The workflow should remain stable because it is organized around responsibilities, not hype cycles.

> **DURABILITY TEST** — **If the model labels change tomorrow, the workflow should still make sense: I build, ChatGPT collaborates, tools assist, Codex may execute later, and I remain responsible.**

02 · THE OPERATING PHILOSOPHY

I am not trying to become the most AI-assisted developer possible. I am trying to become a better software engineer who happens to use AI deliberately.

AI should help me

- Learn unfamiliar concepts without hiding the hard parts.
- Think through project structure, APIs, tests, and implementation options.
- Review code and expose blind spots before they become bugs.
- Debug with better hypotheses and cleaner next steps.
- Document decisions and preserve project context.
- Reduce repetitive friction without replacing creative ownership.

AI should not do for me

- Decide the architecture without my understanding.
- Replace deliberate practice with pasted answers.
- Turn every question into a dependency on a model.
- Automate the parts of software engineering I actually want to learn.
- Create elaborate tooling before a real project needs it.

> **A USEFUL PAUSE** — **Before asking the model, ask: “Am I trying to understand this, finish this, or avoid this?” The answer tells me how much help to request.**

03 · THE WORKFLOW TODAY

The immediate workflow should be simple enough to use every day. It should feel like a strong project room attached to my real development environment, not a complicated agent orchestration system.

VS Code + Terminal · The workbench

Write and inspect the code.
Run builds, tests, benchmarks, and Git commands.
Keep the repository as the source of truth.
Treat executable behavior as more authoritative than any model output.

ChatGPT Plus · The project room

Maintain persistent project instructions and reference files.
Host long-running conversations about the project.
Help with explanations, debugging, design options, test planning, documentation, and learning.
Keep context nearby without forcing me to build custom tooling prematurely.

GPT-5.6 Sol Medium · The serious daily driver

Use for the majority of meaningful software-engineering conversations.
Ask it to teach, tutor, pair program, review, critique, and help plan.
Expect it to handle most workflow tasks without needing a specialized API model.
Treat its answers as proposals to understand and verify, not decisions to obey.

Sonnet API · The future specialist layer

Use later for narrowly scoped utilities, not daily conversation.
Give it selected files, stable prompts, and clear output formats.
Use it for second opinions, structured reviews, and repeatable analysis.
Keep it metered, intentional, and human-triggered.

Codex - later · The bounded executor

Use only when I can define the task clearly.
Prefer branches, tests, summaries, and explicit review gates.
Delegate implementation effort, not engineering responsibility.
Do not rush into it before I can evaluate the resulting diffs confidently.

A typical project session

Step	Action	Purpose
1	Open the repository and ChatGPT Project	Bring together code, context, instructions, and the current objective.
2	Name the next small outcome	A bug fix, feature slice, test target, refactor, experiment, or learning goal.
3	Use GPT-5.6 Sol Medium as collaborator	Ask for explanation, implementation options, debugging hypotheses, review, or test ideas.
4	Write and run the code myself	Keep understanding and executable verification at the center.
5	Capture decisions	Add notes, comments, ADRs, documentation, or issue updates when decisions matter.
6	Escalate later only if recurring friction appears	Build a Sonnet-powered tool only after the need repeats enough to justify it.

04 · CHOOSING THE RIGHT LAYER

The useful question is not “Which model is best?” The useful question is “Which layer fits this kind of work?”

Need	Best layer	Examples
Direct practice	No AI first	Implement a small function, read docs, run a test, trace a compiler error, or sketch the data flow.

Need	Best layer	Examples
Interactive help	GPT-5.6 Sol Medium in ChatGPT Plus	Teach a concept, debug a failure, review a small design, explain an error, plan tests.
Persistent context	ChatGPT Project	Keep instructions, notes, files, and ongoing project conversations in one place.
Repeatable specialist review	Sonnet API tool - later	Review shaders, audit architecture, generate a test plan, summarize project health.
Bounded execution	Codex - later	Create a branch, implement a narrow task, run tests, summarize changes, stop for review.

THE PRACTICAL RULE

Start with direct work or GPT-5.6 Sol Medium. Build API utilities only when a pattern repeats. Reach for Codex only when the task is narrow enough to review safely.

The HOW / SHOULD / DO split

Question type	Meaning	Recommended layer
HOW	I need help understanding or implementing something.	GPT-5.6 Sol Medium, with me writing and verifying the result.
SHOULD	I need help comparing tradeoffs and risks.	GPT-5.6 Sol Medium first; Sonnet API later for structured second opinions.
REPEAT	I keep asking the same review question across projects.	Build a small Sonnet-powered utility.
DO	I can define the task, tests, and acceptance criteria.	Codex later, on a branch, with review before merge.
PRACTICE	The point is to learn by doing.	No AI, or ask for hints instead of answers.

05 · LEARNING DIFFICULT DOMAINS THROUGH PROJECTS

Learning should be project-driven. The project exposes gaps; the model helps explain the gaps; deliberate practice closes them; the next version of the project reveals the next layer.

The model changes roles as I learn

Role	What GPT-5.6 Sol Medium does	What I still own
Teacher	Introduces the domain, vocabulary, prerequisites, and intuition.	Decide what matters and verify with external resources when needed.
Tutor	Asks questions, gives exercises, offers hints, and reviews my solutions.	Do the work, struggle productively, and correct misunderstandings.
Pair programmer	Helps structure code, debug, and generate options.	Write, run, test, and understand the code.
Reviewer	Finds inconsistencies, missing tests, unclear abstractions, and edge cases.	Accept, reject, or adapt the feedback.

Applied example: forecasting software and PDEs

Suppose I want to build forecasting software, but I do not yet understand partial differential equations, numerical methods, stability, or simulation design. The right move is not to ask AI to produce a finished forecasting engine. The right move is to use the project as a curriculum.

PHASE 1 · ORIENTATION

Use GPT-5.6 Sol Medium to map the domain: what forecasting systems contain, where PDEs appear, what math prerequisites matter, and what a minimal version could be.

PHASE 2 · INTUITION

Ask for simple explanations, visual analogies, small derivations, and comparisons between ODEs, PDEs, discretization, and numerical stability.

PHASE 3 · TUTORING

Request exercises, hints, quizzes, and solution review. Ask the model not to give the full answer too early.

PHASE 4 · SMALL SIMULATIONS

Build one-dimensional heat or wave simulations, then two-dimensional versions. Encounter boundary conditions, time steps, grids, and stability in real code.

PHASE 5 · ENGINEERING

Design modules, tests, validation checks, visualization, profiling, and documentation. Keep the system understandable before making it fancy.

PHASE 6 · TOOLING

Only after repeated pain appears, create Sonnet-powered utilities such as simulation-review, numerical-test-plan, or performance-review.

THE LEARNING LOOP

LEARN -> BUILD -> DISCOVER -> STUDY -> REVISE -> EXPLAIN

A good checkpoint is: “Can I explain why this abstraction, method, or test exists?” If not, I should slow down and close the understanding gap before adding more complexity.

06 · PROJECT STRUCTURE AND ENGINEERING JUDGMENT

A large part of the workflow is about getting unstuck at the level between architecture and typing code: files, folders, interfaces, tests, naming, and the first concrete implementation path.

When opening VS Code feels blank

Use GPT-5.6 Sol Medium to convert an architectural idea into a first working skeleton. The goal is not to outsource design; it is to create enough structure that I can start making real engineering decisions.

Project type	Useful model prompt	What I should inspect
Discord bot	“Propose a minimal folder structure for commands, events, config, persistence, and tests.”	Whether the structure matches the bot’s actual complexity.
Visualization tool	“Suggest modules for data loading, transformations, rendering, interaction, and export.”	Whether data flow is clear and testable.
Graphics app	“Help separate rendering, resources, scene state, shaders, platform code, and tools.”	Whether abstractions reflect real boundaries.
API service	“Sketch routes, services, database access, validation, tests, and configuration.”	Whether dependencies point in sane directions.
Simulation project	“Separate numerical methods, scenarios, visualization, validation, and experiments.”	Whether correctness checks are first-class.

THE FIRST STRUCTURE IS DISPOSABLE

A starter structure is a learning scaffold. Once the project reveals its real shape, refactor. Early structure should help motion, not pretend to be final.

07 · INTERNAL TOOLING: EARNED, NARROW, AND USEFUL

The Sonnet API layer should not become “AI everywhere.” It should become a small workshop of tools that solve repeated, well-understood problems.

A good internal tool does four things

- Selects only the files relevant to one task.
- Builds a stable, task-specific prompt.
- Calls the model with explicit output requirements.
- Returns a report that I review and act on.

Good early candidates

Tool	Input	Output
shader-review	Shader source, related tests, target platforms, assumptions	Numerical edge cases, missing tests, portability risks, performance questions.
architecture-review	Selected modules, notes, dependency map	Coupling concerns, tradeoffs, alternatives, and questions worth investigating.
test-plan	Feature code, expected behavior, known risks	Prioritized unit, integration, property, and failure-path tests.
docs-audit	README, ADRs, comments, public APIs	Confusing sections, missing decisions, outdated instructions, better examples.
project-health	Repository summary, selected files, recent changes	Hotspots, maintainability risks, testing gaps, suggested next reviews.

Cost and complexity guardrails

- Do not send the whole repository by default.
- Prefer file selection, summaries, and stable prompt templates.
- Constrain the output format so reports are easy to review.
- Cache stable context when possible.
- Avoid continuous background analysis until there is a concrete need.
- Treat API usage as intentional spending, not an invisible subscription.

> **THE TOOLING PRINCIPLE** **Automate the packaging of good questions, not the abdication of engineering judgment.**

08 · CODEX LATER: EXECUTION WITH GUARDRAILS

Codex is not required for the workflow to work today. It becomes useful later if I can define bounded tasks and review the output confidently.

Good Codex-shaped tasks

- Add tests for documented edge cases.
- Implement a narrow feature with clear acceptance criteria.
- Prepare a branch that refactors a small module without changing behavior.
- Update documentation to match already-merged behavior.
- Run the test suite and summarize failures without guessing fixes.

Bad Codex-shaped tasks

- Own this subsystem.
- Redesign this architecture however you think best.
- Make broad changes and merge them automatically.
- Fix all bugs in this repo.
- Deploy without a human review gate.

Automation ladder

Level	Description	Move up when...
0	Manual practice and direct understanding.	I need to learn or the work is small.
1	Model suggestions: tests, docs, commit messages, review notes.	The model improves clarity but I still do the work.
2	Deterministic scripts: formatting, builds, benchmarks, checks.	The same manual steps repeat reliably.
3	Sonnet-powered review utilities.	The same analysis prompt repeats across files or projects.
4	Codex bounded execution.	I can specify, test, and review the task safely.
5	Delivery automation with controls.	CI/CD, rollback, auditability, and staging are mature.

HEALTHY DELEGATION

“Create a branch, add tests for these edge cases, run the suite, summarize failures, and stop for review.”

09 · FINANCIAL GUARDRAILS

The financial plan should be boring. Pay for the workspace that is useful every day. Meter the specialist layer later. Avoid building an expensive AI operating system before real projects justify it.

BASELINE · PREDICTABLE RECURRING VALUE

ChatGPT Plus as the persistent project workspace.
GPT-5.6 Sol Medium as the serious daily driver.
No need to chase additional model access before the workflow proves itself.

FUTURE METERED USAGE · USAGE-BASED VALUE

Start Sonnet API usage only when a tool has a clear job.
Set a small monthly budget before experimenting.
Track what each utility actually saves or improves.
Increase spending only after repeated value is obvious.

API tool checklist

- Does this tool solve a repeated problem?
- Could GPT-5.6 Sol Medium in ChatGPT handle this interactively instead?
- Does the tool send only necessary files?
- Is the output constrained and easy to review?
- Is there a human-triggered reason to run it?
- Would a deterministic script be better?

10 · IF AI DISAPPEARED TOMORROW

The tools would change. The workflow would survive.

AI-assisted role	Non-AI replacement
Interactive teacher and tutor	Books, courses, lectures, exercises, professors, study groups.
Architecture critic	Mentors, design reviews, RFCs, communities, prior art.
Code reviewer	Peers, maintainers, static analysis, test suites.
Utility layer	Scripts, linters, deterministic analysis, documentation habits.
Executor	Manual work, automation scripts, careful collaborators, review processes.
Project memory	ADRs, README files, notebooks, issues, diagrams, commits.

WHAT REMAINS

Curiosity, deliberate practice, project-based learning, testing, documentation, review, ownership, and the joy of building difficult things.

11 · THE COMPACT PLAYBOOK

PROTECT CURIOSITY.
Do not let speed replace understanding.

KEEP THE PROJECT CENTRAL.
Models and tools serve the work, not the other way around.

USE GPT-5.6 SOL MEDIUM AS THE DAILY SERIOUS COLLABORATOR.
It is enough for most learning, building, debugging, reviewing, and planning.

USE SONNET API ONLY WHEN A TOOL EARNS IT.
A utility should solve a repeated, narrow problem with selected context and a reviewable report.

TREAT CODEX AS LATER INFRASTRUCTURE.
Use it only when bounded execution, tests, branches, and review gates are natural.

EARN COMPLEXITY.
Build first. Automate recurring pain later.

REVIEW EVERYTHING IMPORTANT.
Generated code and analysis are proposals until verified.

PRESERVE OWNERSHIP.
Delegate effort, not responsibility.

BUILD A DURABLE WORKFLOW.
The philosophy should survive model and vendor changes.

FINAL STATEMENT

I am the principal engineer. ChatGPT Plus is my collaborative project room. GPT-5.6 Sol Medium is my serious daily collaborator. Sonnet API may become my specialist utility layer. Codex may become a bounded executor later. My projects remain the center. My curiosity and judgment remain mine.

THE OPERATING PROMISE — **LEARN DEEPLY. BUILD INTENTIONALLY. AUTOMATE DELIBERATELY.**

PROTECT THE PARTS OF SOFTWARE ENGINEERING THAT YOU LOVE.