

A FIELD GUIDE FOR WORKING AUTHORS

Five ways to automate a novel

Choose the setup that fits your writing.

By Stacey M Anderson | Updated October 29, 2026

Chat windows. Coding agents. Third-party apps. Editor extensions. Software you build yourself. Every one of these can be a useful writing setup. Each also brings work you might not expect.

I have used all five. This guide draws on those experiences to help you choose where AI belongs in your process, what you want it to do, and how much infrastructure you want to maintain.

There is no ladder.

These are five ways to arrange the work, not five levels of author sophistication. You can use chat to explore a character, an agent to check continuity, and an app to organize the manuscript. You do not have to graduate from one to the next.

Two choices shape the result

THE MODEL	THE HARNESS
The model doing the writing affects voice, instruction-following, reasoning, context capacity, refusals, and cost.	The interface and tools around the model affect what it can access, which actions it can take, and how you save and repeat the work.

The boundaries overlap. A chat product may include file tools and agent features; an editor extension may host a coding agent. Treat the five lanes as useful descriptions of workflows, not permanent limits on products.

Originally prepared September 2, 2026. This edition updates capability and cost claims and uses the current Stacey M Anderson branding. Historical examples remain labeled as such.

START HERE / THE TWO DIALS

When the output is wrong, investigate the setup.

The model and the harness work together. The same model can produce different results when you change its instructions, supplied material, tools, settings, or review process. Swapping the model can change the result even when the rest stays the same.

Same model, different harness

A chat can work from uploaded chapters, project knowledge, saved instructions, and supported connected sources. An agent with file access can read and update a working folder. In either case, the useful question is what the model actually received for this task, not what exists somewhere in the account.

A draft followed by critique and revision is a different workflow from a single generation. More passes create opportunities to catch problems, but they can also introduce new ones. Check the revised scene against the original purpose.

Same harness, different model

Models differ in cadence, metaphor, dialogue, instruction-following, and how they handle long or difficult material. Prompting and examples can influence those tendencies without guaranteeing a particular voice. Compare on your own scenes, with the same brief and review standard.

A practical diagnostic

Wrong facts or continuity? Check whether the relevant source was supplied, current, retrieved, and understood. Then check the instructions and the model's reasoning. A factual error alone does not identify which part failed.

Right facts, flat prose? Review the brief, examples, hidden product defaults where documented, and revision process. Then compare another model using the same material. Voice is affected by more than the model name.

A refusal or missing action? Check the provider's rules, enabled tools, permissions, and model behavior. Changing the interface does not automatically change all of those constraints.

Change one thing you can explain. Keep the input and output so you can tell whether the change helped.

This is a working framework drawn from practice, not a controlled comparison or a ranking of products. Product features and plan limits change; test the exact setup you intend to use.

LANE 01 / CHAT & PROJECTS

A place to think with the story.

A conversational workspace is often the quickest way to explore a premise, question a scene, or work through a revision. Projects can keep reference files and instructions together. That makes chat useful for ongoing work as well as one-off questions. [1, 2]

What it does well

A short path to useful work. You can start with a scene and a concrete question instead of designing a whole production system.

Visible creative decisions. Discuss alternatives, compare passages, and challenge advice before accepting a rewrite. This is where many of my genre, tone, and heat-level conversations begin.

Reusable context. Project instructions, uploaded files, and saved prompts can reduce repetition. Keep your own portable copy of important instructions and story facts.

Convenient access. Web and mobile interfaces can make small tasks easy to revisit. Availability still depends on the product, connection, and tools you need.

What it asks of you

Manage the source of truth. An uploaded copy may differ from the manuscript you edited elsewhere. Decide which file is current and how accepted revisions get back into it.

Check what was actually read. Context and retrieval limits still matter. Supplying a whole book does not prove every detail was used correctly. Work in purposeful sections and verify continuity.

Build repeatability deliberately. Save successful instructions and examples. A useful conversation can become a reusable asset; it does not become one merely by staying in your history.

Inspect the available actions. Direct folder editing, batch work, and simultaneous checks depend on the product and enabled tools. Some chat experiences include agent capabilities; others require more manual transfer.

From my work

Many saved workflows began as conversations that worked once. Chat did not become obsolete when I started using agents. It became one part of a larger process: a good place to ask whether a scene is doing the job I intended.

If chat gives you the access and control your task needs, it can be the right finished setup.

LANE 02 / CODING AGENTS

Repeatable work against real files.

An agent with the right tools can read a manuscript folder, follow a saved workflow, revise files, and run checks. Coding is one use of that arrangement. Structured writing work is another.

What it does well

Durable working memory. Chapters, plans, ledgers, and instructions can live in files the agent reads when needed. This reduces reliance on one long conversation, though retrieval and context management still require care.

Reusable workflows. A procedure can be saved, versioned, tested, and applied to another book. The result still needs judgment; a repeated process does not guarantee identical prose.

Coordinated checks. Where supported, separate passes can inspect continuity, timeline, voice, and drift. Parallel work needs clear roles and a way to reconcile conflicting edits.

Portable materials. Plain files are easier to back up and move. Some workflow behavior remains tied to a particular tool, model, or plugin, so moving is not always cost-free.

What it asks of you

A learning curve. Desktop and editor interfaces can reduce terminal work. You still need to understand files, permissions, checkpoints, and what the agent changed.

Usage management. Agents may use a subscription allowance or metered API billing. Both can impose limits. Confirm the active billing route and set appropriate caps before a long run. [3, 4]

Workflow maintenance. Instructions and helper tools can go stale. Consistent folders and a current manuscript ledger matter more as the volume grows.

Time to read. Faster drafting can move the bottleneck to reviewing and finishing the output. Count that attention as part of production.

From my September 2026 notes

One recorded novel run reached 36 chapters and 85,966 words, with continuity and drift checks and revisions flowing into earlier setups and later plans. That is an example of orchestration, not a capability exclusive to a particular interface. A separate July run hit a spending cap: a lesson about its billing setup, not proof that every agent is metered.

The lasting benefit is a process you can inspect and improve on the next book.

LANE 03 / THIRD-PARTY TOOLS

An interface someone else maintains.

A purpose-built writing app can give you a story codex, chapter organization, prompt tools, and model access without requiring you to build them. Other tools provide a general interface or a local model runtime. These are related options, not interchangeable products.

What it does well

Useful structure immediately. You can test a working interface before investing months in your own. A codex with a clear editor may be enough for the book you are writing.

Less engineering work. The vendor or community handles much of the interface development. That can be valuable even when you pay a subscription.

Model choice where supported. Some tools use their own bundled models; others accept your API key or connect to multiple providers. Check which models and controls the specific tool exposes.

Reusable community knowledge. Templates and presets can offer a starting point. Inspect their assumptions and adapt them to your voice and process.

What it asks of you

Fit your method to a structure. The app's data model influences how you store story facts and plan chapters. Try a representative chapter before moving a whole series.

Test the exit. Export a small project early. Check whether prose, notes, metadata, and links survive in formats you can use elsewhere. Migration effort varies widely.

Check automation rather than assuming it. Review loops, agents, batch operations, and integrations differ by product. Do not assume every writing app is a one-shot generator, or that every app can automate your entire business.

Count the full cost. A subscription, API use, hardware, and your checking time may all matter. An open-source interface does not make its hosted model calls free.

A note on local models

A fully local setup can reduce external data transfer and avoid hosted per-token charges. Verify the actual data flow: connectors, telemetry, and remote APIs can still send data out. Hardware, electricity, maintenance, model licenses, and model behavior still apply. Local does not mean unrestricted or automatically better for your prose.

From my work

Typing Mind served a real purpose for me as one interface for several models. I later needed a different workflow. That is a fit decision, not a reason every author should leave a tool that works.

LANE 04 / EDITOR EXTENSIONS

An interface over files you control.

An extension can add manuscript navigation, buttons, forms, and AI tools inside an editor. It may work over plain files or use its own storage. Check the actual design before assuming your project is portable.

What it does well

Build on an existing application. You inherit an editor, file tree, settings, and extension system. That can reduce the work of building a writing interface.

Keep manuscript files accessible. An extension designed around Markdown or other open formats can leave the prose easy to back up and use elsewhere. Other apps can offer this too.

Combine interface and agent work. An editor can host both the writing surface and a coding assistant. The five lanes overlap here in a useful way.

Package a workflow for others. Buttons and guided steps can make a personal procedure easier to teach. Distribution still requires documentation, installation checks, and support.

What it asks of you

Learn the host editor. A familiar interface for a developer can be an obstacle for a novelist. Test the first-use experience with the people you want to help.

Test away from your own machine. An extension may accidentally depend on files or tools already installed on your computer. A clean installation test can reveal those assumptions.

Maintain dependencies. Editor APIs, extensions, and model services change. A free editor does not guarantee free model use or a maintenance-free workflow.

Check portability and access. Desktop extensions may not work in a browser or on a tablet. Web and remote options exist, but support depends on the editor and extension.

From my September 2026 notes

An audit found a teaching extension that worked on my machine but lacked content needed elsewhere. Another early build claimed eight sidebar icons; the rule that replaced it was one extension, one icon. These were design and packaging lessons, not reasons to abandon the approach.

A small interface can make a good workflow easier to use. It still has to work on the reader's machine.

LANE OS / YOUR OWN SOFTWARE

Fit the system to your work.

Custom software can connect pen names, catalogs, manuscripts, and publishing tasks in ways other tools handle awkwardly. Build when the specific gap justifies the upkeep.

What it does well

Represent your actual process. You choose the data structure and the actions the interface makes easy. That flexibility is useful when you can explain what existing tools fail to do.

Connect the work. A catalog, planning board, writing surface, and export workflow can share information and reduce manual transfers.

Support the devices you design for. A responsive hosted app can work away from the desk. It needs appropriate authentication and accessible design; being custom does not provide those automatically.

Give you control over priorities. You can choose which feature matters next. Keeping portable data and documented exports makes that control more durable.

What it asks of you

An ongoing software job. Hosting, security, backups, dependencies, testing, and support compete with writing time. A working prototype is the beginning of maintenance.

Dependence on other systems. You control your code, but hosts, APIs, browsers, and model providers can still change. Build recoverable workflows and a way to export the data.

Scope discipline. A new theme can feel more rewarding than finishing a difficult writing feature. Decide what the tool must help you do before adding another surface.

A real exit decision. Sunk cost can make retirement difficult. The question is whether the next month of maintenance earns its place in your publishing work.

From my work

My September notes recorded nearly fifty app deployments, a 52-article manual, and a catalog bridge across five stores. They also recorded twelve themes before the writing surface was finished: useful integration alongside scope creep.

By October 26, I had retired LifeOS and my personal Cloudflare Tunnel setup. My current direction is cloud-hosted work. Owning a tool includes deciding when to stop maintaining it.

Build for a specific gap, with a maintenance budget you are willing to keep paying in time.

THE CHOOSER / SEVEN QUESTIONS

Choose by what the work needs.

01 / What must the AI read or change?

Uploaded reference files may be enough. If it must update your working manuscript, check direct editing, permissions, version history, and how you recover an unwanted change.

02 / Will you repeat this task?

Save the brief, examples, and success criteria. Choose a project, prompt library, skill, or workflow that you can inspect and revise.

03 / Do you need several checks at once?

Look for orchestration and independent review passes. Agents, extensions, and apps may support them; test the exact setup and reconcile overlapping changes.

04 / Where do you need to work?

Try the real task on your phone, tablet, or desktop. A mobile chat interface does not guarantee that every file operation or agent action is available there.

05 / Will someone else use this workflow?

A shared project or written procedure may suffice. Build an extension or app when the interface solves a real teaching or usability problem.

06 / What are your privacy and content requirements?

Check data handling, retention settings, provider policies, licenses, and the model's behavior. Test representative material before choosing a hosted or local setup.

07 / Which step is actually slowing you down?

Measure drafting, checking, revising, organizing, and publishing. Automate the costly step you found, then measure the complete process again.

A small trial beats a theoretical perfect setup. Use one familiar chapter, preserve the original, and include the time needed to check the result.

AT A GLANCE / WORKFLOW FIT

Compare the tradeoffs.

Typical patterns, not guaranteed product features. Verify the exact product, plan, configuration, and export path you intend to use.

SETUP	CONTEXT & REPEATABILITY	REVIEW & ACCESS
Chat & projects	Uploads, project instructions, saved prompts, and supported connections. Keep manuscript copies synchronized.	Sequential review is common; agent features vary. Web/mobile access is often convenient.
Coding agents	Working files plus reusable instructions and tools. Needs reliable retrieval and checkpoints.	Can coordinate checks where supported. Desktop, editor, web, or remote access varies.
Third-party tools	Codex, prompt library, and file handling depend on the app. Test export early.	Review loops and integrations vary. Use a representative task to verify them.
Editor extensions	Often work over local files; storage and workflow support depend on the extension.	Can host agent tools. Web, remote, and mobile compatibility are separate questions.
Your own software	You design the data and process, including recovery and export.	You build or integrate review and device support, then maintain it.

SETUP	COSTS TO COUNT	COST OF LEAVING
Chat & projects	Plan fees or limits; manual transfers and checking.	Export conversations and files; preserve reusable instructions.
Coding agents	Subscription allowance or API charges; setup and maintenance.	Files may be portable; tools and workflows may need adaptation.
Third-party tools	Subscription, bundled credits, API fees, or local hardware.	Depends on export completeness and proprietary formats.
Editor extensions	Editor/extension fees if any, model use, setup, support.	Plain prose may move easily; interface behavior may not.
Your own software	Development, hosting, model use, security, and ongoing upkeep.	You own the migration, retirement, and data-export work.

THE NEXT STEP / KEEP THE AUTHOR

The bottleneck moves.

In my own work, faster drafting created a larger reading queue. Better organization helped me see the queue, but it did not make the final judgment for me. Reading, revising, finishing, and deciding what earns publication still took attention.

The September 2 catalog snapshot recorded **3,008,768 words across 1,183 chapter files in 73 book folders under ten pen-name prefixes**. Those prefixes were folder labels, not a current count of active fiction pen names. These are historical working files, not a claim about published output or current inventory.

The lesson I keep returning to is simple: find the slow step, choose the smallest useful intervention, and inspect the result. The machinery has to earn its place in the writing day.

Keep these five distinctions

Chat can give you a useful space for thinking and drafting with reusable context.

Agents can make procedures actionable against working files.

Third-party tools can save interface-building work when their structure fits.

Extensions can bring a focused interface and automation into an editor.

Custom software can solve a specific fit problem while making you responsible for its upkeep.

Use the setup that helps you write, judge, and finish the book. Change it when the work changes.

Sources & edition notes

Capability references checked October 28, 2026. Feature availability varies by plan and rollout. These references support the capability and billing corrections; they are not independent verification of my historical project examples.

[1. OpenAI: Projects in ChatGPT](#)

[2. Anthropic: What are projects?](#)

[3. OpenAI: Using Codex with your ChatGPT plan](#)

[4. Anthropic: Use Claude Code with your Pro or Max plan](#)

Original guide developed with Claude assistance from my workflows in September 2026. Content revised October 28; branding updated October 29, 2026. A practical field guide, not a controlled research study.

[Read the companion article at StaceyMAnderson.com](#)