OSIIX PUBLIC DRAFT V0.15

Operational Design Guide for Safer AI-Assisted Development

A working method for turning AI speed into changes, verification, and records that people can manage in practice.

VERSION

Public Draft v0.15

CREATED

2026-08-01

UPDATED

2026-08-01

AUTHOR

mars70

SUPPORT

AI-assisted editing

VALIDATION

GPT-5.5 / Codex CLI v0.154.6

OVERVIEW

About This Guide

This guide summarizes methods I tried while using AI for development and operational support to reduce drift and rework. It is not a general standard or the only correct approach; it presents one practical way to use AI in real work.

In my environment, it became much easier to reconstruct the current state when I defined in advance what AI could handle, what could be changed, what had been verified, and what should happen next.

> **Purpose:** To present a practical operating example for carrying out AI-assisted development in a form that people can review, correct, and resume.

Official primary source: The version published in the OSIIX Library is the primary public source for this guide. When referring to redistributed or similar materials, verify that the original publisher is the OSIIX Library.

Authoring and review environment

This guide summarizes observations from mars70's personal hands-on experience. It is not based on analysis of model internals or hidden technical mechanisms. The structure, wording, layout, and prompt examples were developed with assistance from OpenAI GPT-based tools available as of August 1, 2026, including ChatGPT GPT-5.5 Thinking and Codex CLI v0.154.6.

The approval template, VUN reporting format, and Codex prompt examples in this guide are public-facing adaptations of operating patterns considered primarily with GPT-5.5 Thinking and Codex CLI v0.154.6 as of August 1, 2026. They are not guaranteed to behave identically with a particular model, future versions, or third-party tools. Because AI products change quickly, Created and Updated dates are shown.

About version references: The ideas described here—such as defining the change boundary, recording verification results, and separating production operations—are not limited to one particular model. However, prompt examples and operational details may need to be adjusted for the model or tool being used. The listed versions are reference information for the environment in which this guide was created and reviewed.

About the names used in this guide: R0–R5, NOW.md, HANDOFF_PROMPT.md, and Verified / Unverified / Next are examples of classifications and names used in the author's own environment. They are not general standards. Adopt the underlying ideas and adjust names, the number of stages, and the file structure to suit your environment.

Disclaimer and usage notes

This guide is not professional advice on law, security auditing, certification, incident response, or production operations. Any work involving real system changes, deployment, DNS, mail, authentication, firewalls, personal information, or customer data must follow the applicable responsibilities, contracts, laws, and approval procedures of the relevant organization, individual, or group.

Reuse and quotation

CONTEXT

1. Why This Operating Method Is Needed

AI works very quickly when producing initial code from a short description, classifying logs, or drafting documentation. But when an existing repository, an active server, multiple configuration files, past decisions, and published documentation are involved, simply asking it to “fix things appropriately” is risky.

The problem is not that AI always makes mistakes. The problem is that, when the work remains broadly scoped, it becomes unclear which information is

authoritative, which files may be changed, and what has actually been verified. The longer a conversation continues under those conditions, the more explanation accumulates while the deliverable itself stops moving forward.

Approach to avoid

Letting AI continue making changes through a long conversation without first defining the goal, target files, forbidden areas, and verification conditions.

Preferred approach

Break the work into small units, use Git as the source of truth, and define the change boundary and verification conditions before handing the task to AI.

01

Limit the task to one goal

02

Define the change boundary

03

Confirm the source of truth

04

Verify

05

Record and close out

AUDIENCE

2. Intended Readers and What This Guide Does Not Cover

Intended readers

- People using AI-assisted tools such as ChatGPT or Codex to work with existing repositories
- Small teams and individual developers who want to keep Git and documentation in sync
- People who have experienced gaps between AI output, implementation, verification, and records
- People who want to organize AI use as a sustainable working method rather than a temporary trick

Not covered

- Methods for fully automating development with AI
- Security guarantees for a specific service
- A substitute for legal advice, audits, or certification
- Step-by-step procedures for changing a production environment

What follows is one example of methods I have tried for using AI in real work. Use only the parts that fit your own environment.

PRINCIPLES

3. Six Principles for AI-Assisted Development

In my environment, keeping the following six ideas in view made it easier to turn AI output into actual deliverables. They do not need to be adopted in exactly the same form, but it was useful to connect approval, change boundaries, the source of truth, verification, and records.

01

Small approval units

Limit each task to a size a person can review.

02

Explicit write set

Decide in advance which files may be edited.

03

Git as the source of truth

In my environment, GitHub remote main and the local worktree—not the chat—are the reference point.

04

Documentation sync

Update the necessary documentation in the same unit as the implementation.

05

Verification log

Record what was verified and what was not.

06

Synchronized closeout

Close out changes, verification, documentation, commits, and the next action at the same boundary.

BOUNDARY

4. Small Approval Units

When a broad objective is handed to AI as-is, investigation, implementation, verification, documentation updates, and publication work become mixed in one conversation. Mixed work is harder to review and harder to roll back when something fails.

For that reason, I divide approvals into units that make both “what we will do this time” and “what we will not do this time” explicit. For example, read-only investigation, docs-only updates, implementation, runtime changes, and publication work are approved separately.

Practical guideline: If the change target, verification method, and completion criteria cannot be explained at a glance within one approval, the task is too large.

Poor example

```
Do everything required to publish this feature.
```

Better example

```
This time, update only docs/NOW.md and docs/CHANGELOG.md. The goal is to
record the previous verification results and make the next SINGLE_ACTION
clear. Do not touch server, Nginx, DNS, mail, or deploy settings.
```

WRITE SET

5. Explicit Write Set

The write set is the list of files AI may edit for the current task. Without it, a request such as “fix whatever is necessary” may lead AI to modify surrounding files with good intentions.

The write set is not merely a task instruction; it is a safety boundary. Deciding which files should change makes Git diff review shorter and helps reveal unexpected side effects.

```
Approved write set:
- docs/NOW.md
- docs/CHANGELOG.md

Do not edit:
- public/**
- AGENTS.md
- README.md
- server/runtime/DNS/mail/deploy settings
```

Read set / write set / areas not to touch

For publication or server-related work, separate not only the files that may be edited but also what may be read and what must not be touched.

Category	Meaning	Example
Readable scope (Read set)	Areas that may be read for verification	docs/NOW.md, docs/CHANGELOG.md, app/config.py
Editable scope (Write set)	Areas that may be edited in this task	docs/NOW.md, docs/CHANGELOG.md
Areas not to touch (Forbidden set)	Areas that must not be changed or operated on in this task	Nginx, DNS, mail, systemd, and deployment settings

Material treated as secret

.env files, SSH private keys, API keys, access tokens, settings containing passwords, customer information, and personal information are not merely prohibited from being changed. As a rule, their contents should also not be displayed, pasted into chat, written to logs, or summarized.

SOURCE OF TRUTH

6. Git as the Source of Truth

Conversations with AI are convenient, but they are not suitable as the source of truth. They contain interim hypotheses, outdated assumptions, failed ideas, and unverified recollections.

In my environment, I determine the current state from GitHub remote main and the corresponding local worktree. Before and after work, I check HEAD, origin/main, remote refs/heads/main, and the worktree state. When using GitLab,

Bitbucket, self-hosted Git, or feature branches, replace these with the remote repository and baseline branch agreed for the project.

```
git fetch origin main
git status --branch --short
git rev-parse HEAD
git rev-parse origin/main
git ls-remote origin refs/heads/main
```

When I complete work that is applied directly to main, I confirm `HEAD == origin/main == remote refs/heads/main` and `worktree clean` For feature-branch or pull-request workflows, confirm that the current working branch matches its corresponding remote branch.

DOCS SYNC

7. Update Documentation Together with Implementation

If code is changed while documentation is postponed, the next AI or person may read outdated instructions and begin the wrong work. In my environment, gaps between implementation and documentation directly became incorrect assumptions in the next session.

More documentation is not automatically better. I assign distinct roles to documents and avoid repeating the same information.

Document	Role	What it should contain
NOW.md (the author's current-state file)	Current state	Current status, next action, unverified items, and prohibited boundaries
CHANGELOG.md	History	Confirmed changes, dates, commits, and verification results
GOAL_AND_ROADMAP.md	Goals and stages	Goals, policy, completion criteria, and stages
HANDOFF_PROMPT.md (the author's handoff file)	Handoff	Minimum information needed by the next worker
AGENTS.md (the author's working-rules file)	Working rules	Persistent rules for AI or Codex

The important point is not to add more explanation, but to leave enough information that the next task can begin without confusion.

> **Note:** The document structure shown here is one example from my own environment. The important point is not to use the same filenames, but to make the roles of current state, history, handoff, and working rules explicit.

VERIFICATION

8. Verification Logs

A statement such as “verified” does not show what was actually checked. A verification log should record the commands run, the targets checked, the results obtained, and the areas not verified.

```
Checks run:
- git fetch origin main
- git status --branch --short
- git diff --check
- Markdown fence / HTML comment / Jinja scan
- git rev-parse HEAD
- git rev-parse origin/main

Result:
- PASS
- HEAD == origin/main
- worktree clean
```

The purpose of a verification log is not to make AI’s explanation trustworthy, but to leave a state that a person can inspect later.

Caution (avoid context bloat): Pasting full command output or large logs into chat fills the screen and consumes model context, which can reduce accuracy or break the conversation. A practical approach is to show the executed command, a summary, and a PASS/FAIL result in chat while storing detailed logs in local files or dedicated log storage.

CLOSEOUT

9. Close Out Implementation and Documentation Together

Closeout is the process of finalizing the state at the end of a task. When implementation, verification, documentation, commits, and the next action are left at different points in time, it becomes unclear what is complete and what remains unfinished.

A synchronized closeout records the changes, changed files, verification results, unverified items, commit, and next action as one unit.

```
Closeout:
- Result: PASS
- Changed files:
  - app/example.py
  - docs/NOW.md
  - docs/CHANGELOG.md
- Checks run:
  - pytest
  - git diff --check
  - parser-safety scan
- Commit: abc1234
- Final state:
  - HEAD == origin/main
  - worktree clean
- Next single action:
  - R4_PUBLICATION_READONLY_VERIFY
```

IMPROVEMENT 1

10. Separate Work Types and Risk Levels

Using the same process for a minor documentation update and an operation that affects production can either create excessive review or leave important approvals

missing. I therefore use R0–R5 (the author's example classification) to make work types and risk levels easier to distinguish.

This classification is not a general standard; it is an example from my own workflow. The important point is not the names R0–R5, but clearly distinguishing work with different characteristics, such as read-only work, documentation-only work, implementation, and production operations.

Rank	Name	Allowed work	Requires separate approval
R0	Read-only / Survey	Read, inventory, and organize the current state	File edits, Git operations, and runtime changes
R1	Docs-only	Update only approved documents	Code edits, published-content changes, and runtime changes
R2	Diff plan	Define the change plan, write set, and verification conditions before implementation	Implementation itself, deployment, reload, or restart
R3	Implementation	Implement and verify within the approved write set	Unapproved files, runtime changes, or publication
R4	Closeout / Publication prep	Close out verification results, documentation, commits, and publication preparation	Production publication, DNS, mail, and firewall operations
R5	Runtime / Deploy	Explicitly approved production or runtime work	Additional changes or broader rollout outside the approved scope

How This Classification Helped

- It avoids excessive preliminary work for simple tasks.
- It prevents high-risk work from proceeding under a lightweight approval.
- It reduces the chance that AI expands the scope during the task.
- It communicates the weight of a task quickly during handoff.

Handling Production Operations

Operations that affect production environments or external services should not be executed solely on AI's judgment. A person should review the target, procedure, impact, stop conditions, and rollback method, and the work should proceed only within an explicitly approved scope.

In my current workflow, a person makes the final execution decision and performs the operation. In this guide's example classification, this kind of work is labeled R5 for convenience.

IMPROVEMENT 2

11. Create an Approval Template and Reuse the Same Format

When every request is written freely from scratch, the goal, change boundary, prohibited scope, or verification method may be omitted. I therefore created an approval template containing the necessary fields and reuse the same format.

The following is one example used in my environment. You do not need to adopt the same field names or structure. What matters is deciding which fields your environment needs—for example the goal, change boundary, areas not to touch, verification method, and stop conditions—and then using them consistently.

```
WORK_ID:
WORK_RANK:
GOAL:
SOURCE_OF_TRUTH:
REQUIRED_READS:
APPROVED_WRITE_SET:
FORBIDDEN_SET:
RESTRICTED_SECRET_MATERIAL:
ALLOWED_COMMANDS:
REQUIRED_CHECKS:
STOP_CONDITION:
DOCS_TO_UPDATE:
COMPLETION_CRITERIA:
OUTPUT_FORMAT:
NEXT_SINGLE_ACTION:
```

Template Example from My Environment

```
WORK_ID: R1_DOCS_ONLY_CLOSEOUT_SYNC_YYYYMMDD_01
WORK_RANK: R1_DOCS_ONLY
GOAL: Record the latest verification results in docs/NOW.md and docs/
CHANGELOG.md.
SOURCE_OF_TRUTH: GitHub remote main and matching local worktree.
REQUIRED_READS:
- docs/NOW.md
- docs/CHANGELOG.md
APPROVED_WRITE_SET:
- docs/NOW.md
- docs/CHANGELOG.md
FORBIDDEN_SET:
- public/**
- server/runtime/DNS/mail/deploy settings
- unrelated files
RESTRICTED_SECRET_MATERIAL:
- .env
- SSH private keys
- API keys and access tokens
- files containing passwords, customer data, or personal data
ALLOWED_COMMANDS:
- git status / git diff / git diff --check / git rev-parse
REQUIRED_CHECKS:
- changed files exactly match approved write set
- Markdown parser-safety scan
- final HEAD == origin/main if commit/push is in scope
DOCS_TO_UPDATE:
- docs/NOW.md
- docs/CHANGELOG.md
COMPLETION_CRITERIA:
- verified facts and unverified items are recorded
- NEXT_SINGLE_ACTION is one item only
OUTPUT_FORMAT: closeout summary only
NEXT_SINGLE_ACTION: R2_DIFF_PLAN_FOR_NEXT_CHANGE
```

Benefits

- AI is less likely to expand the goal on its own.
- The review target becomes clear.

- History can be recorded in a consistent format.
- The template is easy to reuse as a prompt for Codex.
- Missing approvals or verification conditions are easier to identify before work begins.

IMPROVEMENT 3

12. Record Verified / Unverified / Next Separately

A major risk in AI-assisted development is that an unverified assumption gradually becomes treated as a verified fact. To prevent this, separate Verified, Unverified, and Next at the end of every work report.

Item	Meaning	What to record
Verified	What was actually verified	Commands run, files inspected, and results obtained
Unverified	What has not yet been verified	Runtime not checked, deployment not performed, external service not checked, and similar items
Next	Exactly one next action	Next approval unit, next work ID, and next verification boundary

```
Verified:
- docs/NOW.md and docs/CHANGELOG.md were updated.
- Changed files match the approved write set.
- Markdown parser-safety scan passed.
- HEAD == origin/main after push.

Unverified:
- Runtime state was not checked.
- Nginx reload/restart was not performed.
- DNS/mail/server settings were not changed.

Next:
- R2_DIFF_PLAN_FOR_PUBLICATION_ROUTE
```

Why Separate Them

Separating these three items distinguishes observed facts, unverified scope, and the next action. This makes it easier for the next worker to judge what can be trusted and what must not yet be treated as fact.

> Verified / Unverified / Next is not merely a format for shortening AI explanations. It is a safeguard against mixing verified facts with unverified assumptions.

PRACTICE 1

13. Stopping and Rollback When Work Fails

AI-assisted development should not assume that failures will never happen. It should assume that work can be stopped safely when failure occurs. Because problems can appear even after an item has been marked Verified, define the stopping procedure in advance.

State	Stop condition	Safer response
The same error continues	Two or more similar fixes do not improve the result	End the conversation and restart with only Observed facts and Unverified items
The write set must be expanded	Editing an unapproved file becomes necessary	Stop work and obtain additional approval
Verification is not possible	The reproduction command or expected result is unknown	Do not continue implementation; return to the R2 Diff plan
A high-risk operation is required	Reload, restart, deployment, DNS, mail, or firewall work becomes necessary	Separate it into an R5 approval
An unintended change is found	Unapproved files, secret material, or unnecessary large changes are mixed in	Save the diff, then let a person decide whether to discard, revert, or redesign

Rollback caution

`git reset --hard` is powerful, but it deletes unsaved changes. For a safer stopping procedure, first save the diff, review the changed files, and clarify how branches or working copies will be handled. Destructive discard operations should be limited to cases explicitly approved by a person.

```
Failure protocol:
1. Stop additional implementation
2. git status and git diff --stat to inspect the change scope
3. Save the diff if necessary
4. Record Verified / Unverified / Risks
5. A person decides whether to discard, revert, or redesign
6. Restart using HANDOFF_PROMPT.md or the closeout format
```

PRACTICE 2

14. Session and Context Management

As a conversation becomes longer, old instructions, failed fixes, and interim hypotheses become mixed together, making it easier for AI to lose track of the current approval boundary. After one approval unit is complete, switch to a new session when appropriate.

Reset trigger	Reason	What to provide when restarting
One approval unit is complete	Do not carry completed context into the next task	Latest Git state, closeout, and Next single action
The same explanation starts repeating	The conversation is probably no longer progressing	Observed facts, current write set, and unresolved errors
AI proposes work in a prohibited area	The approval boundary may have been lost	A short approval template containing FORBIDDEN_SET
Error correction enters a loop	Failed approaches remain in the context	Reproduction conditions, latest diff, and a summary of failed attempts
The chat is slow or long	It is disadvantageous for both cost and accuracy	Minimum restart information using HANDOFF_PROMPT.md

```
Minimum information for restarting a session:
- Source of truth: GitHub remote main + local worktree
- Current HEAD / origin/main
- Approved WORK_RANK and write set
- Verified facts only
- Unverified items
- Next single action
- Forbidden set
```

PRACTICE 3

15. Automating Secret-Leak Prevention

A reminder not to write secret information is not sufficient. Assuming that both people and AI make mistakes, I use mechanisms that detect accidental commits or secret material before publication.

Measure	Purpose	Operational note
Maintain a clear .gitignore	Exclude .env files, private keys, generated files, and local settings from Git	Put examples in files such as .env.example and never include real values
Pre-commit hooks	Inspect formatting, secret material, and unnecessary files before commit	Document setup steps and failure handling
secret scanning	Detect strings that resemble API keys, tokens, or private keys	Manage exception rules because false positives occur
Checks in CI	Detect local oversights before a pull request or merge to main	Do not expose secret information in failure logs
Pre-publication scan	Check that internal names or secret information do not remain in HTML, PDF, or README files	Separate the public version from internal review material

Writing examples for public versions

Do not use real environment paths, IP addresses, internal hostnames, customer names, or values resembling real tokens even in examples. Standardize examples on sample, example, and placeholder.

PRACTICE 4

16. JSON / YAML Approval Templates

If you want to reuse the template in tools or scripts, as I do, you can provide JSON or YAML in addition to Markdown. This is an optional example for readers who need it; Markdown or plain text is sufficient for ordinary chat use.

Role of each key: `forbidden_set` identifies targets that must not be changed or operated on in the current task. `restricted_secret_material` identifies material that, in addition to being protected from modification, should not be read, displayed, summarized, written to logs, or pasted into chat. The Text, YAML, and JSON versions use the same roles.

YAML version

```yaml
work_id: R3_IMPLEMENTATION_EXAMPLE_YYYYMMDD_01
work_rank: R3_IMPLEMENTATION
goal: Implement the approved small change only.
source_of_truth:
  - GitHub remote main
  - Corresponding local worktree
approved_write_set:
  - app/example.py
  - tests/test_example.py
forbidden_set:
  - server/runtime/DNS/mail/deploy settings
  - unrelated files
restricted_secret_material:
  - .env
  - SSH private keys
  - API keys and access tokens
  - files containing passwords, customer data, or personal data
required_checks:
  - pytest
  - git diff --check
  - changed files match approved write set
```

JSON version

```
{
  "work_id": "R3_IMPLEMENTATION_EXAMPLE_YYYYMMDD_01",
  "work_rank": "R3_IMPLEMENTATION",
  "goal": "Implement the approved small change only.",
  "source_of_truth": ["GitHub remote main", "Corresponding local
worktree"],
  "approved_write_set": ["app/example.py", "tests/test_example.py"],
  "forbidden_set": ["server/runtime/DNS/mail/deploy settings", "unrelated
files"],
  "restricted_secret_material": [".env", "SSH private keys", "API keys
and access tokens", "files containing passwords, customer data, or
personal data"],
  "required_checks": ["pytest", "git diff --check", "write set
verification"],
  "closeout_format": ["Result", "Changed files", "Verified",
"Unverified", "Risks", "Next"]
}
```

> JSON and YAML can help AI interpret the instruction, but they are not approvals by themselves. Treat them as machine-readable representations of a scope approved by a person.

PRACTICE 5

17. Review Scope and Anti-Patterns

Small approval units are useful, but if they become too small, waiting for human review becomes a bottleneck. Choose a practical scope that balances safety and progress. The following 15–30 minute range is simply a guideline that I find manageable, not a general standard.

Criterion	Practical scope	Signs the scope is too small or too large
Review time	One approval or pull request can be reviewed in about 15–30 minutes	Many approvals take only a few minutes each / one review covers the entire project
Purpose	Focused on one goal	Multiple features, refactoring, and documentation cleanup are mixed together
Diff	The changed files and reasons can be explained	The intent cannot be followed from the diff
Verification	Required tests and checks are clear	Verification conditions are vague or excessive
Impact scope	A scope that can be rolled back if it fails	Production impact or broader rollout is included implicitly

When the Units Become Too Small

When approval units are made too small, review and explanation can increase while implementation and documentation move further from completion. Small approval units are not intended to stop work; they are a way to steadily build deliverables, verification, and records while keeping the scope reviewable by a person.

CODEX PROMPT

18. Minimal Prompt for Codex

A prompt for Codex or another AI development tool is not better merely because it is long. What matters is the source of truth, approved scope, prohibited scope, verification, stop conditions, and output format.

```
Use the local worktree as the source of truth.
Do not infer current state from chat history.

WORK_RANK: R3_IMPLEMENTATION
GOAL: Implement the approved small change only.
APPROVED_WRITE_SET:
- app/example.py
- tests/test_example.py
- docs/NOW.md
- docs/CHANGELOG.md
FORBIDDEN_SET:
- server/runtime/DNS/mail/deploy settings
- unrelated refactors
RESTRICTED_SECRET_MATERIAL:
- .env
- SSH private keys
- API keys / access tokens
- files containing passwords or personal data
REQUIRED_CHECKS:
- tests for changed behavior
- git diff --check
- changed files exactly match approved write set
- parser-safety scan for edited Markdown
STOP_CONDITION:
- stop if write set must expand
- stop if runtime/deploy operation becomes necessary
- stop if restricted secret material must be read or displayed
CLOSEOUT_FORMAT:
- Result
- Changed files
- Verified
- Unverified
- Risks
- Next
```

NOTICE

19. Usage Notes and License

This public guide organizes practical ideas for carrying out AI-assisted development more safely. It reflects observations from the author's experience and does not guarantee compliance with any particular product, law, or security standard.

Usage Notes

Any work involving real system changes, deployment, DNS, mail, authentication, firewalls, personal information, or customer data must follow the applicable responsibilities, contracts, laws, and approval procedures of the relevant organization, individual, or group.

License

Disclaimer

This guide is not legal advice or professional advice on security audits, certification, incident response, or production operations. It has been generalized to avoid including internal information, but application to any environment is at the user's own judgment and responsibility.

AI assistance disclosure: This guide summarizes observations from mars70's personal hands-on experience. AI assistance was used for structure, wording, and layout. It includes operating patterns considered primarily with GPT-5.5 Thinking and Codex CLI v0.154.6 as of August 1, 2026, but does not guarantee behavior with a particular model or future version.

SUMMARY

20. Summary

The important question in AI-assisted development is not whether AI is used. It is how much is delegated to AI and where people draw the boundary.

In my environment, small approval units, an explicit write set, Git as the source of truth, documentation sync, verification logs, and synchronized closeout made it easier to turn AI's speed into actual deliverables.

Adding stopping procedures, session management, secret detection, JSON/YAML templates, and review-scope guidance also makes it easier to address problems that emerge after the workflow is in use.

These methods are not the only correct answer, but they serve as useful landmarks when AI-assisted work becomes lengthy and the current state is no longer clear.