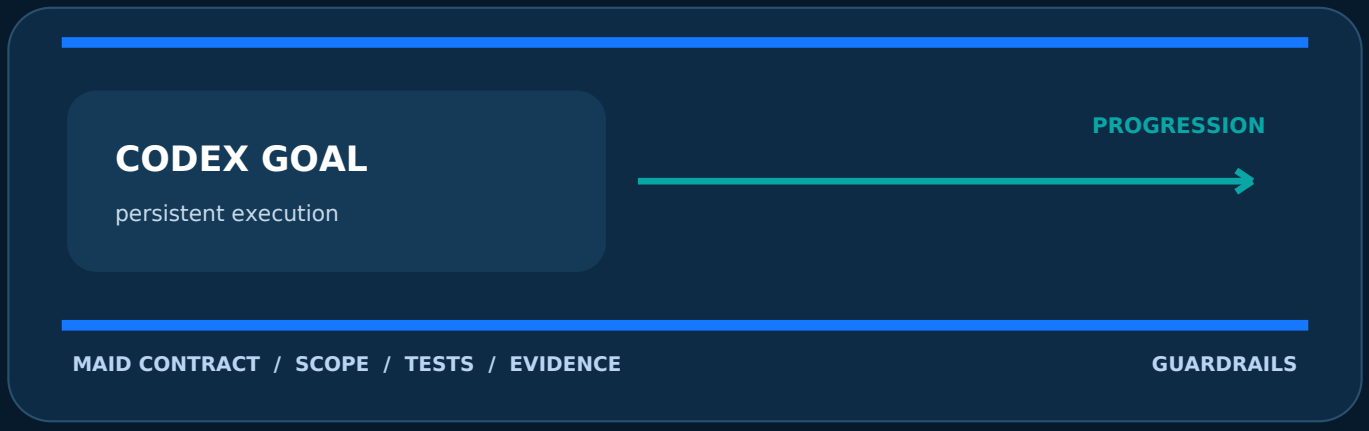
AI-DRIVEN CODER / FIELD GUIDE

Prepared Autonomy

How I Use MAID Runner and Codex Goals to Turn Pre-Planned Work into Disciplined, Low-Drift Software Delivery



Codex provides persistent execution.
MAID provides the contract and guardrails.

Plan now. Execute when AI capacity is available. Validate everything.

Mamerto Fabian Jr.
AI-Driven Coder

aidrivencoder.com/MAID

PUBLICATION EDITION | JULY 2026

01 / THE OPERATING IDEA

Prepared autonomy

I separate the hard judgment of planning from the long, repetitive work of implementation.

The moment it clicked

I have increasingly used Codex for development work involving MAID. During a recent period when my available Codex usage refreshed or expanded more generously, I saw the leverage in having substantial development work already planned.

I did not need to spend that execution window deciding what to build. When the right capacity is available, I give Codex a persistent `/goal` to process a queue of approved, implementation-sized contracts until it reaches a real decision boundary.

A personal trigger, not a product promise

Temporary usage behavior made this workflow's value more obvious to me. I do not treat that experience as a permanent OpenAI policy or plan entitlement.

What I prepare in advance

- Epic planning records for work that still needs decomposition.
- Implementation-sized child manifests, ordered by dependency.
- Behavioral expectations and test paths.
- Declared files, artifacts, interfaces, and validation commands.
- Known dependencies, risks, and temptations to avoid.
- Approval, plan-lock, review, and completion evidence.

The operating model

1. Prepare bounded, verifiable work.
2. Let Codex process the ready queue.
3. Let MAID enforce the approved contract.
4. Pause on ambiguity, authority, risk, or failed gates.

Prepared autonomy

Autonomous in progression, constrained in scope, and evidence-driven at completion.

02 / THE CONTRACT LAYER

MAID is infrastructure, not another agent

MAID Runner is an open-source, tool-agnostic Python toolchain that checks code against declarative YAML manifests.

One manifest, six jobs

PLANNING
What are we actually building?

CONTRACT
What did we approve before code existed?

SCOPE
Which paths may change?

ARTIFACTS
Which public structures must exist?

VALIDATION
Which commands prove progress?

AUDIT
What was the agent authorized to deliver?

Three complementary validation streams

1 WHAT

ACCEPTANCE

Behavioral tests define externally observable outcomes.

2 SKELETON

STRUCTURAL

Declared files, classes, functions, components, interfaces, and signatures are checked.

3 HOW

UNIT + IMPLEMENTATION

Conventional tests protect internal correctness and regressions.

A guardrail system, not magic

Structural validation alone does not prove correctness. I combine structure, behavior, tests, review, and completion evidence. MAID does not replace engineering judgment, product judgment, or security review.

03 / THE MANIFEST HIERARCHY

From planning inventory to durable evidence

EPIC PLANNING RECORD
larger map; never implemented directly

CHILD DRAFT MANIFESTS
mutable inventory of bounded units

SELECTED CHILD + RED EVIDENCE
behavioral tests fail for the intended reason

REVIEW + APPROVAL + PLAN LOCK
the pre-implementation contract is sealed

PROMOTED ACTIVE MANIFEST
approved contract enters the active set

BOUNDED IMPLEMENTATION
edits remain inside declared scope

VALIDATED + REVIEWED RESULT
tests, gates, and review support completion

OUTCOME + REUSABLE LESSONS
durable evidence informs future work

What changes at each level

1. **Epic**
   A feature, migration, remediation program, or other multi-stage map. Split it before implementation.

2. **Child draft**
   A mutable planning inventory item for one implementation-sized cycle. Not yet an active contract.

3. **Promoted manifest**
   The approved contract in the active set. Do not silently rewrite it to match the code.

4. **Outcome**
   Completion metadata added after validation and review: status, evidence, findings, and reusable lessons.

The integrity rule

A draft can evolve. A promoted, locked contract must be revised explicitly. The point is to preserve evidence that the approved expectations existed before implementation.

Supported promotion

Use `maid manifest promote manifests/drafts/<slug>.manifest.yaml`. The command migrates the plan lock, red evidence, and self-referencing validation paths. Do not manually copy the draft.

04 / END-TO-END WORKFLOW

Plan, prove red, approve, lock

The first half turns a broad objective into one implementation-ready contract.

CAPTURE → SPLIT → TEST → REVIEW → LOCK

1 Capture the larger goal

Start from a feature, bug, refactor, migration, audit finding, or product objective.

2 Create an epic when needed

Map boundaries, dependencies, risks, and likely child work. Never implement the epic directly.

3 Split into child drafts

For each bounded unit, declare files, artifacts, tests, commands, dependencies, and forbidden shortcuts.

4 Select one child

Process dependency order. Avoid simultaneous agents in overlapping files.

5 Retrieve applicable lessons

Use learn, recall, or insights. Digest relevant evidence; reject stale lessons. Recall never expands scope.

6 Write behavioral tests first

Describe observable behavior, not source strings, private helpers, or a preferred implementation trick.

7 Confirm the red phase

Run tests before implementation. Missing behavior is valid red; a broken dependency or environment is not.

8 Validate and review the plan

Check that the contract is coherent, bounded, specific, and hard to game with a hollow workaround.

9 Approve and lock

Seal the manifest and behavioral tests before production implementation. Legitimate changes require an explicit revision.

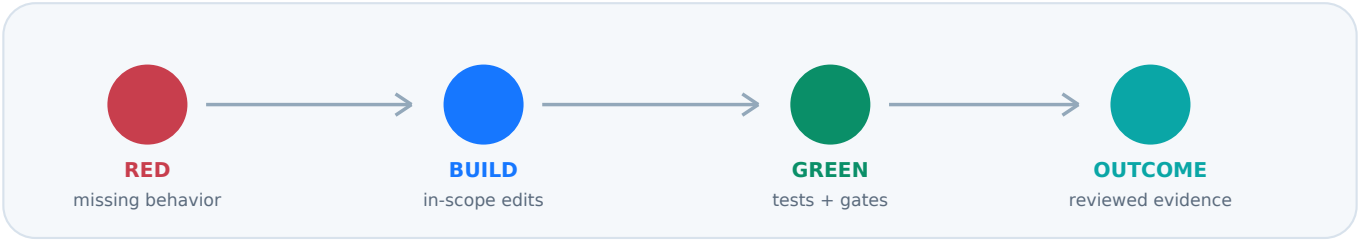
Evidence-bearing commands

```
maid recall --for-manifest
manifests/drafts/<slug>.manifest.yaml --plan-packet
maid validate manifests/drafts/<slug>.manifest.yaml
--mode behavioral
maid plan lock manifests/drafts/<slug>.manifest.yaml
```

04 / END-TO-END WORKFLOW

Promote, implement, review, learn

The second half produces a reviewed result, durable evidence, and the next ready unit.



10 Promote the selected child

Use the supported promotion command so the active path and lock evidence evolve together.

11 Implement within declared scope

Start the task pointer and edit only authorized paths. Pre-edit scope checks can reject undeclared files.

12 Run tests and validation

Execute declared tests, implementation validation, type checks, lint, builds, browser tests, or equivalent project gates.

13 Run implementation review

Look for regressions, security and privacy risk, scope drift, missing tests, architectural problems, and contract gaming.

14 Fix valid findings

Treat review as a gate. Rerun every check affected by the correction.

15 Capture an Outcome

After review, add final status, evidence, findings, and reusable lessons to the manifest's outcome section.

16 Refresh the learning index

Run maid learn so future plans and reviews can recall the completed Outcome.

17 Continue to the next child

Codex can select the next ready contract, or stop cleanly when a genuine blocker requires a human decision.

Reference gate sequence

```
maid manifest promote manifests/drafts/<slug>.manifest.yaml maid task start
manifests/<slug>.manifest.yaml
maid test --manifest manifests/<slug>.manifest.yaml maid validate
manifests/<slug>.manifest.yaml --mode implementation
maid verify --summary --require-plan-lock --require-red-evidence --base-ref
<parent-branch> maid learn
```

05 / CODEX GOAL MODE

Persistence without permission inflation

Goal mode supplies continuity. The manifest supplies the contract, checkpoints, and stopping rules.

Documented Codex behavior

In the desktop app, Codex CLI, and IDE extension, `/goal` sets a persistent goal. The goal text becomes the first prompt and completion criteria. The same task can be steered, paused, resumed, edited, or cleared.

Boundary that remains

Starting a goal does not grant broader access. Existing sandbox and approval policies still apply, and Codex pauses when it needs a decision. Goal mode does not approve commits, pushes, deployment, credentials, or destructive actions.

Verified against OpenAI's current Codex manual: Long-running work and /goal reference.

COPY / ADAPT

```
/goal Process every approved, implementation-ready MAID child manifest in dependency order, one at a time.

For each child:
1. Retrieve and digest relevant completed Outcome lessons.
2. Confirm the manifest is narrow, coherent, and implementation-sized.
3. Ensure behavioral tests exist and fail for the intended reason.
4. Run behavioral validation and required plan review.
5. Confirm approval and plan-lock evidence.
6. Promote the child through the supported MAID command.
7. Implement only within the declared scope.
8. Run the declared tests, implementation validation, and changed-scope verification.
9. Perform an independent implementation review when available.
10. Fix valid findings and rerun affected checks.
11. Capture an evidence-backed Outcome.
12. Refresh the MAID learning index.
13. Continue to the next ready child.

Pause and report clearly if requirements are ambiguous, a security or privacy risk appears, required authority
is missing, the requested change falls outside manifest scope, validation cannot be satisfied honestly, or
human approval is required.

Never weaken tests, rewrite the contract to match an incorrect implementation, conceal a failed check, or
commit and push without explicit authorization.
```

06 / WHY THE COMBINATION WORKS

One objective, two control systems

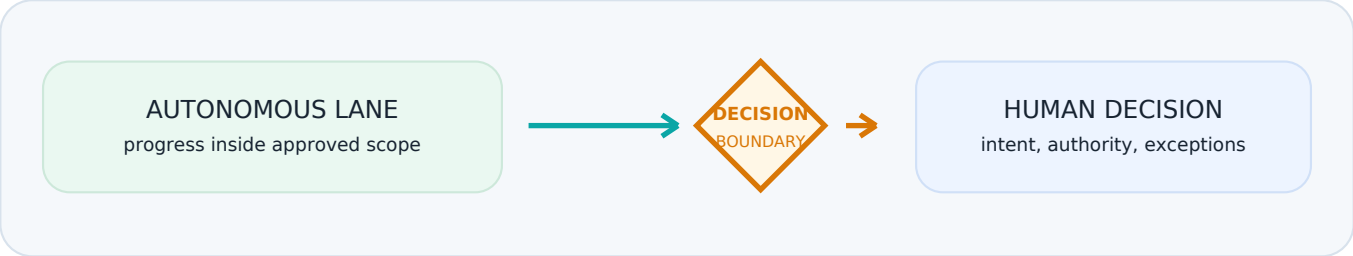
Who owns what

Layer	Primary responsibility
MAID manifests	Intent, file scope, artifacts, constraints, and done criteria.
Behavioral tests	Observable behavior plus red/green evidence.
MAID Runner	Deterministic validation, scope checks, plan locks, and completion gates.
Codex Goal mode	A persistent objective across multiple implementation cycles.
Claude Code (optional)	Another MAID-compatible planner, implementer, or independent reviewer.
Cursor (optional)	IDE editing, inspection, browser-assisted verification, and visual checks.
Git + human approval	Accountability for commits, pushes, releases, destructive actions, and product decisions.

07 / AUTONOMY BOUNDARIES

What 'Almost Autonomous' Actually Means

The aim is controlled progression, not unrestricted freedom.



The agent may autonomously

- Select the next ready child in dependency order.
- Read the approved contract and recalled lessons.
- Run tests, validation, and honest retry loops.
- Make in-scope code changes.
- Review work and capture evidence.
- Proceed to the next ready child.

Codex keeps the work moving.

MAID keeps it honest.

Stop or escalate

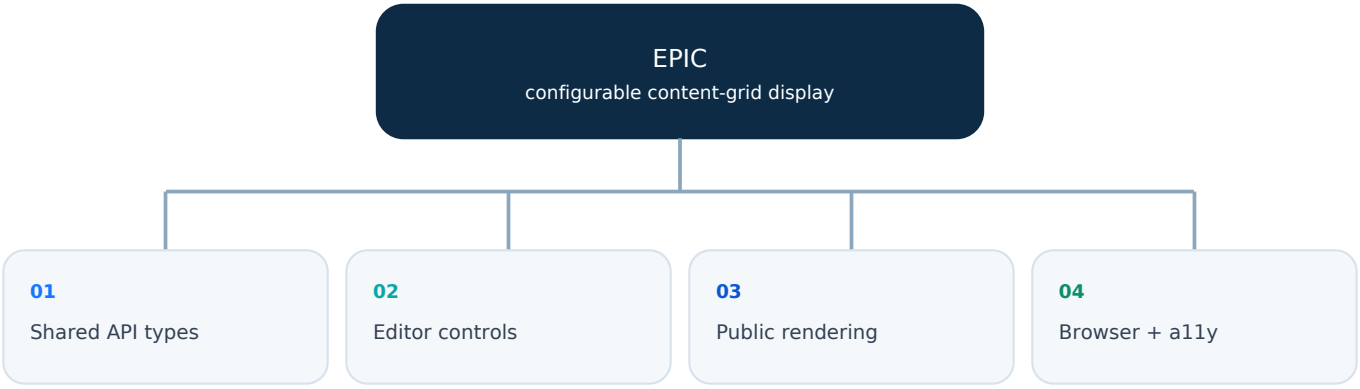
- Requirements are ambiguous or the draft is not ready.
- An epic has not been split into bounded child work.
- Implementation needs undeclared files or a legitimate contract change.
- Security, privacy, authentication, or data exposure risk appears.
- Dependencies, credentials, or the environment are broken.
- A commit, push, deploy, destructive action, or external decision needs approval.

The goal is not 'remove the human.'

Move human attention from repetitive implementation supervision to intent, approval, exceptions, and final judgment.

08 / PRACTICAL EXAMPLE

A configurable content-grid option



How Codex processes it

- Each child is a draft manifest with declared paths, artifacts, behavioral tests, dependencies, and validation commands.
- I approve one implementation-ready child. Codex promotes and executes it before selecting another.
- Tests, implementation validation, changed-scope verification, visual checks, and review must pass before Outcome capture.
- The completed Outcome becomes advisory evidence for later children.

The critical stop

Undeclared type change detected

Suppose the rendering child requires a shared type change that its manifest did not authorize. The scope check stops the edit. Codex does not quietly widen the task.

Explicit evolution

I revise the contract with evidence or prepare the prerequisite manifest. Then validation and approval happen again. The exception becomes visible instead of disappearing into the diff.

09 / TRADEOFFS

Useful because it is constrained

Prepared autonomy shifts cost toward explicit planning and evidence - and away from drift and expensive rework.

What I gain

- Plan before agent capacity is available.
- Turn child manifests into a machine-readable work queue.
- Continue across many implementation steps with one persistent goal.
- Reduce scope drift through narrow, explicit contracts.
- Preserve red-phase and plan-lock evidence so silent test rewriting is detectable.
- Catch missing or structurally incorrect artifacts deterministically.
- Retain accountability and organizational memory through review and Outcomes.
- Move between Codex, Claude Code, Cursor, and other tools without changing the contract.
- Recover from interruptions because durable state lives in the repository, not only in chat history.

Practical leverage

The queue is ready before the opportunity to execute it. I can use scarce attention for decisions, not repeated re-explanation.

What it still costs

- Strong manifests and behavioral tests require upfront work.
- Weak tests can still permit weak implementations.
- Structural validation does not prove business correctness.
- Exploratory work may need discovery before contracting.
- Large or overlapping manifests reduce autonomy and raise risk.
- Agents still need sandboxing, permissions, review, and escalation.
- More validation can consume more tokens and compute, even when it reduces rework.
- No workflow guarantees bug-free software.

A candid boundary

MAID can make deviations visible and completion evidence stronger. It cannot manufacture a good product requirement or replace engineering judgment.

Tool-agnostic by design

Install repository guidance with `maid init --tool codex`, `claude`, `cursor`, `windsurf`, or `generic`. The agent can change; the manifest remains the contract.

FINAL INVITATION

Try one bounded change.

Do not begin with a giant autonomous backlog. Choose one feature or bug fix with observable behavior, narrow file scope, and tests you trust. Prepare the contract, prove red, lock it, implement it, and inspect the evidence.

Plan now. Execute when capacity is available. Validate everything.

MAID Runner resources

- **Source repository**
  https://github.com/mamertofabian/maid-runner
- **Documentation**
  https://maid-runner.readthedocs.io/en/latest/
- **Python package**
  https://pypi.org/project/maid-runner/
- **Introductory video**
  https://youtu.be/0a9ys-F63fQ

Source integrity

Workflow claims and command forms were checked against the public MAID Runner repository and its current draft-manifest, plan-lock, Outcome, and agent-skill documentation. Codex Goal behavior was checked against OpenAI's current official Codex manual, including Long-running work and the /goal command reference. Personal practices and interpretations are presented as my workflow, not as OpenAI policy. MAID Runner is independent open-source infrastructure and is not affiliated with or endorsed by OpenAI.

OpenAI source: https://learn.chatgpt.com/docs/long-running-work

Publication URL: https://aidrivencoder.com/MAID

Mamerto Fabian Jr. / AI-Driven Coder