Software Factory

Version Nov 2026 · verified 2026-11-17

A production line for code — labels as state, a decision model for triage, zero-trust agents, independent review, humans merge.

Contents

Mental model 3

The production line 3

Labels as the state machine 5

Triage with a decision model 7

Zero trust for agents 9

  Protect the pipeline itself 11

Add a planning agent 11

Build with four rules 13

Independent review 15

Merge: humans decide 17

Measure, then widen 18

Further reading 19

Mental model

A software factory is a production line for code: agents do the repeatable steps of delivery, people set the rules and approve what ships. In comes a support email or a GitHub issue; out comes a pull request with green CI and a completed independent review. GitHub labels are the state, GitHub Actions are the conveyor belt, and every rule that matters lives in versioned code and config — never only in a prompt. A person keeps exactly one step: the merge.

The production line

Stage	Who	Hands off with
Gate	Classifier	A clean, de-identified issue
Sort	Decision model + code	`agent-ready` or a side exit
Plan	Planning agent	Plan comment, then `planned`

Stage	Who	Hands off with
Build	Any coding agent	PR with `Fixes #N`
Review	Separate reviewer	Findings back to the builder
Merge	Your team	Merged PR, closed issue

Each stage reads one label and writes the next, so any stage can be swapped — a different coding agent, a different reviewer — without touching the others. Review findings loop back to the coding agent until CI is green; only then does a person see the PR. The gate in front keeps how-to, billing and spam emails out, and strips personal data before anything becomes a public issue.

```
email/issue -> gate -> sort -> agent-ready
  -> plan -> planned -> build -> PR
  -> review <-> fix (loop) -> green
  -> person merges
```

> **Note:** Start with one repository and one narrow issue type. The line pays off through its rules and metrics, not its agent count.

Labels as the state machine

Label	Set by	Next step
`agent-ready`	Triage	Plan request
`needs-human`	Triage or a stuck agent	A person picks it up
`unclear`	Triage	Reporter replies, re-triage
`duplicate`	Triage	Link the original, close
`planned`	Action, on plan comment	Coding agent claims
`agent:working`	Coding agent	PR opened, or timeout

The first four are triage's only outputs, and only `agent-ready` ever reaches an agent. `needs-human` covers auth, secrets, payments, data changes, big fixes, low model confidence and suspected injection. A stale claim — `agent:working` for hours with no PR — goes back to `planned` automatically, so a crashed agent never strands an issue.

```
# swap state in one call: never two at once
gh issue edit 123 \
  --remove-label planned \
  --add-label agent:working
```

> **Gotcha:** An issue carrying two state labels gets picked up by two stages at once. Have the Action that adds a state label remove every other state label in the same step.

Triage with a decision model

Question	Options
Kind	bug, feature, how-to, billing, spam
Info	complete, missing repro, missing expected
Area	auth, payments, data, UI, API, docs
Size	trivial, small, medium, large
Duplicate / Injection	none, or yes (with `#N`)

Ask all six closed questions in one call to a model that returns a probability for every option. Your code, not the prompt, turns the answers into a label, so thresholds are reviewed and versioned like any other code. Post every answer and its probability as an issue comment: humans can audit a decision and tune thresholds from real data.

```ts
type Answer = { option: string; p: number };
type Q = "kind" | "info" | "area" | "size"
  | "duplicate" | "injection";
function route(t: Record<Q, Answer>) {
  const o = (q: Q) => t[q].option;
  const human = Object.values(t)
    .some((a) => a.p < 0.8)
    || o("injection") === "yes"
    || /auth|payments|data/.test(o("area"))
    || o("size") === "large";
  return human ? "needs-human"
    : o("duplicate") !== "none" ? "duplicate"
    : o("info") !== "complete" ? "unclear"
    : "agent-ready";
}
```

> **Gotcha:** Don't parse a free-text verdict from an LLM. If you must use one, force structured output with enumerated options, and calibrate its confidence against a labelled set of 100–300 past issues before trusting a 0.80 cut-off.

Zero trust for agents

In reach	Out of reach
Read the repo	Production secrets, API keys
Push an `agent/*` branch	Merge, force-push
Open a PR, comment, label	Repo settings, branch rules
Package registries	Any other outbound host

Run every agent in a fresh container — rootless Docker, gVisor or a Firecracker microVM — destroyed after each run, with only the rights its step needs. Prefer a GitHub App over a personal token: its installation tokens are

scoped per repository and expire after one hour. Then an agent that obeys a hidden instruction reaches no secrets, merges no code and cannot send data anywhere.

```yaml
# workflow default: least privilege
permissions:
  contents: read
  issues: write
  pull-requests: write
```

> **Warning:** Issue text is untrusted input. An issue can hide orders for an agent in an HTML comment, alt text or a linked file. Ask triage whether the text gives orders to an AI agent, route every “yes” to a person, and never run `pull_request_target` workflows that check out PR code — that hands fork code your repository secrets.

Protect the pipeline itself

- Rulesets on the default branch: require a PR, a human approval and status checks; block force pushes; keep agent identities off the bypass list.
- CODEOWNERS on `.github/workflows/`, `AGENTS.md`, review config and sensitive paths (`auth/`, `billing/`, migrations).
- Pin third-party Actions to a full-length commit SHA.
- Cap tokens, wall-clock time and retries per issue; log every transcript and command for audit.

Add a planning agent

Trigger	Result
`agent-ready` added	Action posts a plan request
Planner runs (read-only)	Plan written as an issue comment
Plan comment by planner bot	Action adds `planned`

The planner reads the issue and the codebase and writes the files to change, the steps and the risks. The coding agent starts from that plan, not the raw issue. It is also the cheapest moment to catch a security problem: the plan names risks the issue never mentioned, before any code exists. Spend your strongest model here — a bad plan wastes every step after it.

```
## Plan for #123
**Goal:** one testable sentence
**Files:** path — why (max ~10, else split)
**Steps:** 1. … 2. … 3. …
**Tests:** what each new test proves
**Out of scope:** what not to touch
**Risks:** authz, input, secrets, PII, data
**Confidence:** high / medium / low
```

> **Gotcha:** Check the comment's author, not just its text. If the Action adds `planned` to any comment starting `## Plan for`, anyone can forge a plan and steer the coding agent.

Build with four rules

Rule	Why
Take only `planned` issues	Triage and planning already ran
Claim with `agent:working`	Two agents never take one issue
Follow the plan	Scope stays reviewable
Open a PR with `Fixes #N`	Merge closes the issue

Use any coding agent — Claude Code, Codex, a cloud agent or your own on a server. Put the rules in `AGENTS.md` and every agent that reads it follows them. One container, one branch, one issue: no state carries between tasks. Agents write a failing test first, run the

same checks as CI before pushing, and after a few failed attempts comment what they tried and relabel `needs-human`.

```
# AGENTS.md
1. Work only on issues labelled `planned`.
2. Add `agent:working` first; drop it on PR.
3. Follow the plan comment. Wrong plan?
   Comment why and stop.
4. Open a PR with `Fixes #N`.
- Test: `npm test`; lint: `npm run lint`
- Never edit .github/, AGENTS.md, CODEOWNERS
- Never delete, skip or loosen a test
```

> **Gotcha:** A `concurrency` group holds at most one running and one pending run; a newer pending run replaces the older one. Use a group per issue (`issue-${{ github.event.issue.number }}`), not one global group, or a claim for one issue can cancel the queued claim for another.

Independent review

Setting (`.coderabbit.yaml`)	Does
`auto_review.auto_incremental_review`	Reviews every new push
`path_instructions`	One rule per folder
`pre_merge_checks.custom_checks`	Your own pass/fail check
`tools.gitleaks` / `tools.semgrep`	Secrets, static analysis
`tools.osvScanner`	Vulnerable dependencies

A separate reviewer reads every push to an agent PR, without the writer's assumptions — ideally a different

model or vendor. The coding agent answers each comment with a failing test and a fix, or with a reason. Cap the loop at about three rounds, then escalate to `needs-human`.

```
reviews:
  path_instructions:
    - path: "src/api/**"
      instructions: >-
        Validate all input; check authz.
  pre_merge_checks:
    custom_checks:
      - name: "Test tampering"
        mode: "error"
        instructions: >-
          Fail if tests are deleted, skipped
          or assertions loosened.
```

> **Warning:** Never let the coding agent review its own PR. Deleting, skipping or loosening a test is the most common way an agent “passes” CI; back the review check with mutation testing on changed files or a coverage floor on changed lines.

Merge: humans decide

Do	Don’t
Read the highest-priority PR first	Read in arrival order
Read big PRs in layers: data, logic, UI	Read a 2,000-line diff top down
Check the PR against its plan	Approve because CI is green
Keep merge rights with people	Give any agent merge rights

When CI is green and the review is complete, a script posts one line to the team chat. A person reads the PR

and merges it. Agents open PRs faster than people can review them, so human review time is the real bottleneck: rank the queue, keep PRs small, and put a WIP limit on open agent PRs.

```
[P1] #482 Fix duplicate invoice on retry
  3 files +84/-12 · CI green · review done
  risk: payments
```

> **Tip:** Ask “would I be comfortable rolling this back?” before merging. Feature flags, reversible migrations and staged rollout turn a bad agent PR into a five-minute revert instead of an incident.

Measure, then widen

Metric	Healthy direction
Agent PR merge rate	Up
Review rounds per PR	Down
Human review minutes per PR	Down

Metric	Healthy direction
Reverts from agent PRs	Near zero
Cost per merged PR	Down
`agent-ready` triage precision	Up

Roll out in steps, widening only as merge rate holds and reverts stay near zero. A false `agent-ready` costs far more than a false `needs-human`, so start strict and loosen with evidence.

1. Shadow: triage comments but labels nothing; compare with humans.
2. Triage live; people still write all code.
3. Plans only; people code from them and rate the plans.
4. Agents build `trivial` / `small` issues in low-risk areas.
5. Lower thresholds and add areas; review metrics weekly.

Further reading

- AGENTS.md

- CodeRabbit configuration reference
- GitHub Actions secure use reference
- GitHub App installation tokens
- OWASP GenAI Security Project
- DORA metrics