

Demo Evaluation Decision Card

Ask for evidence before treating a convincing demo as a production system.

Ask during the demonstration

1. Which real users and workflows does this support today? Distinguish a demonstration from live use.
2. Name three difficult inputs or edge cases. Show what happens when each arrives.
3. What happens when a dependency is unavailable, slow, or returns conflicting data?
4. How are duplicate requests and conflicting records handled?
5. Who owns this system, and who can explain its data flow end to end?
6. What incidents or recovery exercises have tested it? Ask for dates and evidence; a new system may have exercises rather than operating history.
7. What remains before the intended use is safe: access controls, monitoring, recovery, scale testing, or operational ownership?

Ask for a recovery demonstration

1. Use a safe test environment with agreed limits. Do not break a live customer system to prove a point.
2. Record each unanswered question, its potential consequence, the owner, and the evidence needed. A polished screen is not that evidence.

Decision: proceed to a controlled trial / request evidence / stop. Explain the conditions before proceeding.

Rewrite or Modernize Decision Tree

Choose a change strategy after understanding the system and its dependencies.

Work through these options

1. Retire: no required user, integration, record-retention obligation, or business process still depends on it. Plan data disposition and a safe shutdown.
2. Wrap: the core behavior remains useful, but integration is the obstacle. Add an interface with access controls, validation, monitoring, and ownership.
3. Refactor: the desired changes are bounded and the team understands the behavior. Change small pieces behind characterization tests.
4. Replace gradually: route one workflow at a time to a new implementation. Plan data synchronization, reconciliation, rollback, and the eventual shutdown.
5. Rewrite: assess this when required behavior cannot be achieved through bounded changes, or the platform cannot be operated acceptably. Record alternatives, migration costs, business rules, and transition risks.

Before changing anything

1. Map dependencies with users and operators. Capture important current behavior in tests, including exceptions.
2. Use sanitized test data. Check outputs against business requirements; preserving an existing bug is not the goal.
3. Choose an accountable owner, a reversible first step, success measures, and stop conditions. No option is automatically safe.

Chosen approach and evidence: _____. **Next reversible step and owner:** _____.

Cleanup Crew Role Specification

Assess the ability to understand, stabilize, and improve an inherited system.

Look for demonstrated skills

1. Read unfamiliar code: ask for an example they understood well enough to change. Let them explain how they learned it.
2. Trace data: follow an input through transformations, storage, integrations, and reports. Ask where they verified assumptions.
3. Talk to the business: ask about a user conversation that changed their diagnosis.
4. Handle incidents: look for clear prioritization, communication, escalation, and recovery. Use past examples or a calm simulation, not a hostile interview.
5. Choose an investigation: ask for a first hypothesis, how to test it, and what evidence would change their mind.

Set the role up to succeed

1. Provide authorized access, a system owner, business contacts, a safe test environment, and a clear escalation path.
2. Ask for a system map, prioritized findings, recovery risks, and small verified repairs before a broad rewrite.
3. Agree on the boundaries of the work, availability, and handover. Incident skills can be developed through pairing and practice.

Evidence of fit: _____. **Support or training required:** _____.

New Apprenticeship Checklist

Build judgment through reading, testing, user contact, and supported operational work.

For the engineer

1. Trace one feature in code you did not write, from user action to stored data and response.
2. Inspect a permitted, sanitized dataset. Compare your conclusions with an AI-assisted analysis and investigate differences.
3. Sketch your approach before asking AI for alternatives. Explain why you keep or reject each suggestion.
4. Watch a user perform a real task. Compare their expectations with what the system does.
5. Take a bounded integration or maintenance task with review from an experienced engineer.
6. Shadow an incident or on-call rotation with a mentor and clear escalation support. Do not take sole responsibility before you are ready.

For the manager

1. Mix implementation work with diagnosis, tests, documentation, and conversations with users.
2. Review reasoning and evidence, not just the volume of output. Give time to read and ask questions.
3. Use the reading list in Chapter 6 and Appendix B as material for discussion and practice.

This month I understood and changed: _____. **My reviewer and next practice:** _____.

Daily AI Workflow

A repeatable check before accepting AI-assisted changes.

Run the seven steps

1. Define the task: state the required behavior, constraints, inputs, outputs, and acceptance checks.
2. Generate a small change: give the relevant context and explicit boundaries. Keep credentials and private data out of unapproved tools.
3. Read the diff: understand changed behavior, dependencies, permissions, and error paths. Investigate anything you cannot explain.
4. Test correctness: check the required output and business result, not just valid syntax or a successful run. Include failure and boundary cases.
5. Commit a coherent change: preserve a useful history with a message explaining the purpose.
6. Verify stored data and integrations: use authorized access and safe test environments; avoid exploratory writes to production.
7. Check scope and recovery: separate unrequested refactors, rerun affected checks, and know how to undo the change.

For features that call a model

1. Record model, prompt, retrieval, and supported generation settings. Fixed settings can help repeat a test; they do not guarantee identical answers.
2. Evaluate both format and task correctness. Test untrusted inputs and tool permissions. Keep a human accountable for the release.

Change: _____. **Evidence checked:** _____. **Reviewer / rollback:** _____.

AI Security Red Flags Card

Review code, access boundaries, and model-connected tools before release.

Check the code and its access

1. Check identity and authorization separately. Test that a signed-in user cannot read or change another user's records.
2. Enforce permissions server-side on every protected operation. Use least privilege and deny access that has not been explicitly allowed.
3. Use parameterized queries and validate inputs at boundaries. Check uploads, file paths, URLs, and command execution.
4. Use maintained authentication components and appropriate password storage. Do not invent password hashing or session handling.
5. Remove embedded credentials. Rotate any exposed secret; deleting it from the latest file does not revoke it.
6. Keep secrets and unnecessary personal data out of logs. Define access, redaction, and retention for diagnostic records.

Check AI features

1. Treat retrieved pages, documents, and tool responses as untrusted data. Prompt instructions alone are not access controls.
2. Validate tool arguments and outputs. Enforce permissions outside the model and require approval for consequential actions as appropriate to the workflow.
3. Exercise prompt-injection and misuse scenarios in a safe environment. Combine human review, automated checks, and focused security testing.

Finding / impact / evidence / owner / next action: _____.

Production Readiness Scorecard

Record evidence for the level of use a system must support.

Choose the intended level

1. 0 - Demo: a prepared path works under supervision. It is not evidence of readiness for real operations.
2. 1 - Prototype: an experiment with an owner and a decision date, isolated from production users and data. It does not take customer money or hold irreplaceable records.
3. 2 - Internal tool: a named owner, authorized access, appropriate data protection, visible errors, a continuity plan, and human review support real internal work.
4. 3 - Production: retain the operational safeguards from Level 2; access may include authorized customers. Verify dependency failures, access controls, backups and restore, monitoring, realistic load, documentation, and rollback.
5. 4 - Regulated or business-critical: retain Level 3 safeguards and add controls and evidence appropriate to the consequences, applicable obligations, incident response, and independent review.

Apply the requirements correctly

1. Higher levels retain relevant safeguards, not earlier restrictions on who may use the system. A production system does not inherit the prototype's ban on real users.
2. For AI features, verify evaluations, versioned configuration, tool permissions, monitoring, and rollback of model-related changes.
3. Record missing evidence and risk. A failed requirement does not automatically place the system at the level immediately below. This worksheet is not a compliance certification.

Target level: _____. **Evidence links:** _____. **Gaps / owner / due date:** _____.

Hiring Scorecard

Use consistent questions and evidence to evaluate ownership of real systems.

Rate each dimension with evidence

1. End-to-end curiosity: asks about users, data, constraints, and business outcomes.
2. Failure ownership: explains a specific incident, their contribution, and what changed afterward.
3. Tradeoff fluency: explains the costs and limits of a recommendation as well as its benefits.
4. Legacy judgment: investigates existing behavior before proposing replacement.
5. Maintenance experience: can explain how decisions held up after delivery.
6. AI discipline: describes what they delegate, what they verify, and how they handle uncertainty.
7. Business communication: explains technical consequences clearly to people outside engineering.

Run a fair work sample

1. Use a small unfamiliar codebase and a bounded problem. Explain expectations, permitted tools, and available context consistently.
2. Watch how the candidate forms and tests a hypothesis. Provide reasonable accommodations and assess the work, not speed or interview polish alone.
3. Use a 1-5 scale with written examples: 1 = little evidence; 3 = adequate evidence for the role; 5 = strong relevant evidence. Calibrate requirements to the actual job.

Candidate / role: _____. **Evidence and gaps:** _____. **Decision and reasons:** _____.

30 / 60 / 90 Day Action Plan

Turn the book into a bounded plan for one system or one area of practice.

Days 1-30: understand

1. Leaders: choose one important system, use the executive question card, name its owner, and identify the largest evidence gaps.
2. Engineers: trace a real workflow, review an unfamiliar module, and use the daily AI workflow on bounded changes.
3. Early-career readers: pair with a mentor, read existing code, and explain one feature end to end.
4. Experienced readers: document a system you maintained and a specific problem you inherited and resolved.

Days 31-60: improve

1. Choose one high-value improvement with an owner and a measurable result. Use the audit and readiness checklists to guide the work.
2. Practice restore and rollback in a safe environment. Close a monitoring or documentation gap.
3. Pair across experience levels. Make review, user contact, and supported incident practice part of normal work.

Days 61-90: verify and repeat

1. Compare the system with its starting state. Check evidence, remaining risks, and who can operate it when the usual owner is absent.
2. Record what improved, what did not, and the next bounded priority. Set a review date and an operating budget based on actual requirements.
3. If an urgent incident emerges, follow the recovery tracker immediately. The calendar is a planning aid, not a reason to postpone a necessary fix.

One system / goal: _____. Owner: _____. Measure and next review: _____.

Executive Question Card

Ask for explanations and evidence without needing to review the code yourself.

Ten questions for your technical team

1. Can you draw the system and explain how the parts work together?
2. Where does important data enter, change, and leave the system?
3. What happens when a dependency fails or a request arrives twice? What test demonstrates it?
4. Which part is most fragile, and what are you doing about it?
5. Who can operate and repair it when its usual owner is unavailable?
6. Where is AI used to generate code or make decisions?
7. Who reviewed and tested those parts, and what evidence did they record?
8. Can the team change the code deliberately, or only ask a model to regenerate it?
9. When did engineers last observe the people who use the system?
10. How do you detect incorrect outcomes, rather than only checking that servers are running?

What to do with the answers

1. Look for specific examples, demonstrated controls, and honest limits. "I don't know; here is how I will check" can be a useful answer.
2. Record unknowns with owners and review dates. Prioritize by the harm a failure could cause, not the confidence of the speaker.

Most important unknown: _____. **Owner / evidence / review date:** _____.

Day-Zero Recovery Tracker

Organize the first week of recovery while responding immediately to urgent harm.

Use the days as a sequence, not a waiting period

1. Day 1 - Contain: name an owner, pause nonessential changes where appropriate, assess active harm, and set a communication channel. Preserve evidence where feasible without delaying necessary containment. Involve security and other responsible specialists as needed.
2. Day 2 - Investigate: examine access controls, important data flows, backup recovery, and monitoring. Start with the highest-consequence risks; these are starting points, not the only surfaces to inspect.
3. Day 3 - Inventory: identify AI-generated, human-written, and unknown code. Ask whether the team can modify it, whether tests cover failures, and whether malicious input can reach an unintended system.
4. Day 4 - Decide: choose stabilization, constrained operation, rebuilding, or retirement for each affected subsystem. Check whether a proposed wrapper actually controls the hazard; it is not an automatic safe default.
5. Day 5 - Communicate: state verified findings, uncertainty, business impact, actions, owners, and the next update. Make required notifications on the applicable timetable.

Interpret findings and act

1. A "no" to maintainability or failure testing identifies a gap. A "yes" to dangerous access identifies a different risk. Do not wait for all three answers to be concerning.
2. Avoid a broad rewrite before understanding the system. Make urgent repairs when the risk warrants them, using verified facts, appropriate review, and a recovery plan.
3. Keep a system map, prioritized findings, decision log, and communication record. Review progress daily.

Immediate risk: _____. **Containment / repair owner:** _____. **Next update:** _____.

AI Integration Checklist

Check the problem, operating boundaries, and evidence before and after adding AI.

Before building

1. State the business problem, intended users, and why AI is useful. Identify a simpler baseline for comparison.
2. Define acceptable outcomes, unacceptable errors, and who owns the decision to release.
3. Identify data sources, permissions, retention, and information that must not reach the model.
4. Map integrations and dependencies. Agree on tool permissions and human decision points.

Before release

1. Test representative tasks, edge cases, failure conditions, and hostile inputs. Evaluate correctness as well as format.
2. Review generated code, dependencies, authorization, and data handling. Resolve consequential findings.
3. Version relevant prompts, models, retrieval, and supported settings. Plan cost limits, monitoring, fallback, and rollback.
4. Practice recovery and handover. Name who responds when something goes wrong.

After release

1. Monitor outcomes, errors, cost, latency, and user reports. Review unexpected behavior rather than simply retrying it.
2. Rerun evaluations before consequential changes. Recheck permissions and data sources as the system evolves.

Decision: proceed / remediate / stop. Evidence, open risks, owner, and review date: _____.

AI-Generated Code Audit Checklist

Review a defined change or module and record evidence that another engineer can check.

Understand the change

1. Record the scope, owner, affected workflows, dependencies, and intended business behavior.
2. Read changed code and relevant surrounding paths. Verify assumptions and identify areas nobody can explain.
3. Check dependency names, provenance, versions, and maintenance status. Investigate unfamiliar packages.

Check behavior and boundaries

1. Test normal inputs, invalid inputs, empty data, concurrent requests, retries, and partial failures where relevant.
2. Trace important data through storage and integrations. Check duplicate handling, transactions, migrations, and recovery.
3. Test authentication and object-level authorization. Inspect queries, uploads, secrets, logs, and error responses.
4. For model-connected features, test task correctness, prompt injection, output handling, tool permissions, and escalation.

Make a release decision

1. Record findings by impact and supporting evidence. Assign an owner and a verification step to each required repair.
2. Check monitoring, restore, rollback, and documentation. Have another qualified person review consequential changes.
3. Do not use the number of checked boxes as a substitute for judgment. One unresolved severe issue can be enough to stop release.

Module / change: _____. Findings and evidence: _____. Reviewer / decision: _____.