Keep the reasoning with the change.

An engineering note should let the next maintainer understand a decision without reconstructing the original conversation.

A useful engineering note

Record	What to capture
Context	The affected workflow, constraints and current behaviour
Decision	The chosen change and the alternatives rejected
Evidence	A reproduction, test result or measured observation
Follow-up	The owner and condition that would trigger a review

Describe an observable problem

Start with the input, the behaviour observed and the result needed. A small reproducible example is more useful than a broad claim that a component is unreliable.

Link the note to the implementation

Reference the change, the contract and the verification evidence. Keep the note short enough to update when the system changes.

Write for the next maintainer

Describe the business operation before the implementation. If an invoice was created twice, record the source event, the first successful write and the later retry. Include the identifiers needed to follow that sequence, with customer information removed. A statement such as “the integration is unreliable” gives a maintainer no useful starting point.

Separate observations from assumptions. A timeout proves that the caller did not receive a response in time. It does not prove that the receiving system rejected the write. Record the evidence available and any uncertainty that remains. Link to a reproducible test or trace in a location the team can maintain. An inaccessible screenshot in a private conversation is not a durable explanation.

Explain the choice and its limits

Document the alternatives that were seriously considered and the constraint that ruled each one out. For duplicate invoice creation, the decision might be to store an external reference and reconcile before repeating a write. Explain why a simple retry was insufficient and where reconciliation can still fail. This helps a future engineer change the design without accidentally removing its protection.

Give the decision a review trigger. A change to the provider API, transaction volume or accounting process may make a different approach worthwhile. Record the owner, date and implementation link. Update the note when the implementation changes, or mark it as superseded. Two contradictory decisions should never both appear current.

Check whether the note is usable

A reviewer should be able to follow the affected workflow and identify the evidence that would disprove the proposed explanation. Include a successful example, a failure example and the expected recovery behaviour. State whether the change affects existing records, deployment order, access permissions or rollback.

Before closing the work, ask another engineer to locate the relevant code and reproduce the failure using the note. Record missing prerequisites instead of relying on a verbal explanation. Use sample data that can be shared safely and distinguish commands that inspect a system from commands that change it. A useful note reduces future investigation time. Its value comes from the questions it answers, rather than the amount of documentation produced.

Does every code change need a long document?

No. Reserve a decision note for a meaningful tradeoff or non-obvious constraint. Routine changes can carry their context in a clear description and focused checks.

Read the current resource online

https://cobnex-review.rgcsekaraa.workers.dev/resources/engineering-notes/