Make quality a delivery habit.

Connect acceptance criteria, focused automation and exploratory checks to the risks in each change.

Quality planning worksheet

Risk	Check to plan
Business rule changes	Examples covering boundaries and exceptions
Integration changes	Consumer compatibility and failure responses
Interface changes	Critical journeys across input methods and screen sizes
Release changes	Deployment, observation and rollback rehearsal

Put the check at the useful boundary

Business rules often need focused tests close to the code. Integration behaviour needs contract checks. Browser tests should cover a deliberate set of important user journeys.

Remove low-value duplication

Repeated tests that mirror implementation can make changes expensive without finding different failures. Review the suite as the product evolves.

Test the behaviour the business depends on

Quality engineering starts with the operations that must remain dependable. Identify the inputs, business rules and outcomes for those operations. Include the roles allowed to perform them and the systems that own the resulting records. A broad count of automated tests does not show whether the important behaviour is covered.

Use representative examples to agree expectations with the business. An approval workflow should be tested with a complete request, missing information, a rejected request and a user without approval authority. Record the expected state after each action. These examples can inform unit, integration and end-to-end checks without requiring every test to repeat the entire workflow.

Investigate gaps between environments

A passing local test can conceal differences in permissions, configuration or external dependencies. Identify which checks use simulated services and which exercise a realistic integration. State the limits of each environment rather than presenting all successful runs as equivalent evidence.

For a finance integration, a test double may verify payload construction while a provider test tenant verifies authentication and accepted fields. Neither alone demonstrates the correctness of a complete production reconciliation process. Test failure handling and repeated requests at the relevant boundary. Keep test data suitable for the environment and avoid copying production personal information merely to make an example feel realistic.

Keep the test suite useful over time

When a defect is found, decide which test would have detected it at an appropriate level. A focused regression check is often more useful than another expensive end-to-end scenario. Review unstable tests instead of allowing repeated reruns to become the normal release process.

Make failures understandable. A result should identify the behaviour that changed and provide enough context for investigation. Remove obsolete checks when requirements change and maintain important examples with the business owner. Combine automated evidence with proportionate accessibility, usability and exploratory review. The purpose is confidence in the delivered workflow and its recovery paths, not a large test count or a percentage reported without context.

Is test coverage percentage enough to judge quality?

No. It says little about whether the tests exercise meaningful failures or whether the expected behaviour is correct.

Read the current resource online

https://cobnex-review.rgcsekaraa.workers.dev/galileo/