# Master Test Strategy

**A template to adapt for your organization**

This document states what your organization tests, why, who does it, and how you will know whether it is working. It is the parent document: the integration, regression and acceptance strategies each refine one part of it and should not contradict it.

> ***How to use this template.*** *Everything in italics is guidance for you and should be deleted once you have replaced it. Square brackets mark a value to fill in. Work top to bottom the first time; after that, treat sections 3 and 4 as living documents and revisit them every quarter. Keep it short. A strategy nobody reads governs nothing.*

## 1. Purpose and scope

> *Two or three sentences. If this section runs past half a page, the strategy is trying to be a plan.*

Systems in scope

Systems explicitly out of scope

This document is owned by

Reviewed every

Last reviewed

Out of scope is the more useful half. Naming what you are **not** testing prevents the assumption that somebody else is.

## 2. What we test today

> *An honest baseline. You cannot show improvement without it, and this is the section teams are most tempted to skip.*

| Activity | Who does it | How long it takes per release | Automated today? |
|---|---|---|---|
| Regression testing | [team] | [hours] | [none / partial / full] |
| Smoke testing after deploy | [team] | [hours] | [ ] |
| Integration testing | [team] | [hours] | [ ] |
| User acceptance testing | [team] | [hours] | [ ] |
| Exploratory testing | [team] | [hours] | Not applicable |
| [add your own] | | | |

Total manual test effort per release (person-hours)

Escaped defects in the last quarter

Of those, how many a test could plausibly have caught

Date this baseline was captured

That last date matters. In six months this page is the only evidence you have of where you started.

## 3. Tracing tests to requirements

*The point of traceability is not an audit artifact. It is being able to answer two questions: if this requirement broke, would we know? And if this test goes red, what business capability is affected?*

Record the link in whichever direction your team will actually maintain. One row per requirement is usually lighter than one row per test.

| Requirement / story ID | Business capability | Criticality (H/M/L) | Covered by | Gap? |
|---|---|---|---|---|
| [REQ-001] | [Customer can place an order] | H | [test name or "manual only"] | [yes/no] |
| [REQ-002] | | | | |
| [REQ-003] | | | | |

**With Functionize Studio:** a test built from a plain-language description of the requirement stays readable by the person who wrote the requirement, so the trace can be verified by them rather than taken on trust.

## 4. Critical journeys

*List the flows that would hurt if they broke silently. Most organizations land on ten to twenty. If your list has sixty, you have listed features rather than journeys.*

The test: **if this broke at 2am on a Friday and nobody noticed until Monday, what would it have cost?** Revenue, a compliance breach, a support queue, reputation. If the answer is "very little", it is not critical however visible it is.

| # | Critical journey | What it costs if it breaks silently | Covered today? |
|---|---|---|---|
| 1 | [the money flow — whatever results in you being paid] | | |
| 2 | [permission boundary — and the negative case] | | |
| 3 | | | |
| 4 | | | |
| 5 | | | |

## 5. What we automate, and what we do not

> *Score each critical journey. A candidate that fails the first two columns is rarely worth automating whatever its business value, because the test will not be trustworthy.*

| Flow | Business risk if it breaks (H/M/L) | Run frequency | Deterministic? | Data controlled? | Automate? |
|---|---|---|---|---|---|
| [journey 1] | | | | | |
| [journey 2] | | | | | |
| [journey 3] | | | | | |

**Deterministic** means you can state, before the run, exactly what a pass looks like. **Data controlled** means the test does not depend on a record somebody else might change.

State plainly what stays manual, and why. Declaring this is a mark of a mature program, not a gap in one:

- Exploratory testing — nothing automated finds the problem nobody thought to describe.
- Subjective judgment — is this readable, is the tone right, does this error message help.
- Anything needing a physical action — a card reader, a scanner, a real phone call.
- One-time migration checks that will never run again.
- Behavior still being designed. Come back when it settles.

> **With Functionize Studio:** the run-frequency bar drops sharply, because building a test is a plain-language description rather than a day of selector work. Flows that were never worth scripting become worth describing. The other four columns still apply.

## 6. Test levels and who owns them

| Level | What it proves | Owner | When it runs | Gate? |
|---|---|---|---|---|
| Unit | A component behaves | [dev] | Every commit | Yes |
| Integration | Components agree at the seams | [ ] | [ ] | [ ] |
| System / end-to-end | A user journey works | [ ] | [ ] | [ ] |
| Regression | What worked still works | [ ] | [ ] | [ ] |
| Acceptance | The business agrees it is right | [ ] | [ ] | [ ] |

> Each level should prove something the level below cannot. If two levels would fail for the same reason, one of them is redundant.

## 7. Environments and test data

> Environment used for automated regression

Environment used for acceptance testing

How test data is provisioned

How test data is reset, and how often

Who to contact when an environment is down

> Do not automate against production. You will create real orders and send real emails. And an unstable environment produces failures that teach your team to distrust the whole suite.

> **With Functionize Studio:** named variable sets — execution presets — let the same tests run against different environments without duplicating anything, and credentials live in encrypted variables rather than in a test step.

## 8. Entry and exit criteria

> *Be specific enough that two people would reach the same decision without a meeting.*

| Gate | Entry criteria | Exit criteria |
|---|---|---|
| Build accepted for test | [ ] | [ ] |
| Ready for acceptance testing | [ ] | [ ] |
| Ready for release | [ ] | [ ] |

## 9. Defect management

| Severity | Definition | Response time | Blocks release? |
|---|---|---|---|
| 1 — Critical | [ ] | [ ] | Yes |
| 2 — High | [ ] | [ ] | [ ] |
| 3 — Medium | [ ] | [ ] | No |
| 4 — Low | [ ] | [ ] | No |

Add one rule that pays for itself: **for every defect that escapes to production, ask whether a test could have caught it. If yes, write that test before the incident is closed.**

## 10. How we will know this is working

> *Pick five. Measuring twenty means acting on none.*

| Measure | Baseline (today) | Target | Reviewed |
|---|---|---|---|
| Escaped defects per quarter | [ ] | Falling | Quarterly |
| Critical journeys covered | [ ] | 100% | Monthly |
| Time to feedback after a change | [ ] | Minutes | Monthly |
| Maintenance share of automation effort | [ ] | Under 25% | Quarterly |
| Flake rate | [ ] | Near zero | Weekly |

> Two measures to avoid: **counting tests**, which says nothing about whether you would catch a broken checkout, and **chasing a coverage percentage**, which drives people to write cheap tests. Ask instead: if the whole suite passes, what do we now know is true?

## 11. Roles

| Role | Name | Responsibility |
|---|---|---|
| Strategy owner | [ ] | Owns this document |
| Test lead | [ ] | Day-to-day coverage decisions |
| Automation owner | [ ] | Conventions, tiers, promotion rules |
| Reads a red result | [ ] | Named per scheduled suite — see the regression strategy |
| Business acceptance | [ ] | Signs off acceptance testing |

## 12. Sign-off

Prepared by / date .....

Reviewed by / date .....

Approved by / date .....

Next review due .....