Testing — A Complete Training Reference

Author: Mira Banerjee — QA / Test Lead

Audience: Mohit + anyone building or joining a QA function

Date: 2026-05-18

Status: Living document; revised as tools and prices change

1. Why testing matters (and why we invest in it)

Bugs get exponentially more expensive the later you catch them. The classic numbers from research (IBM, NIST, others) — a bug found in requirements costs $1 to fix. The same bug in design: ~$10. In code: ~$100. In testing: ~$1,000. In production: ~$10,000+ (sometimes much more, especially with security or compliance impact). The whole point of structured testing is to shift the catch point LEFT — earlier in the cycle — where fixes are cheap.

Three categories of cost a bug in production carries that a bug in dev doesn't: (1) customer harm — lost trust, churn, support tickets; (2) operational cost — emergency rollback, oncall pages, incident response; (3) compliance / legal — DPDPA / RBI fines, audit findings, contract breaches. The cost of testing infrastructure is always less than the cost of one serious production incident.

Testing is not a phase. It's a discipline embedded across the whole development cycle. The most expensive way to do QA is to treat it as a 'team that tests stuff after devs are done.'

2. The two big buckets of testing

Every test fits into one of two categories — functional or non-functional.

Category	Asks	Example
Functional	Does the system do what it's supposed to?	Can a parent pay a fee successfully?
Non-functional	How well does the system do it?	Can it handle 10,000 parents paying fees at the same time without slowing down?

3. Functional testing — every type, in order of scope

From smallest scope (one function) to largest (a real user doing a real thing):

3.1 Unit testing

Tests the smallest piece of code — one function or one method — in isolation. No database, no network calls, no UI. If the function takes inputs A and B and should return C, the test verifies exactly that. Thousands of unit tests, each runs in milliseconds.
Owner: developers themselves. Catches: logic errors, edge cases, regressions. Tools: xUnit / NUnit / MSTest (.NET); Jest, Vitest, Mocha (JS/TS); JUnit, TestNG (Java); pytest (Python). All free.

3.2 Component testing

A step up from unit. Tests one component (e.g., one Angular component, one React component, one .NET service class) WITH its immediate dependencies mocked out. Verifies the component behaves correctly when wired up.
Owner: developers. Tools: Angular Testing Library, React Testing Library, Storybook with interaction tests, .NET test fixtures. All free.

3.3 Integration testing

Tests multiple components working together. Real database (test DB), real services. Catches the wiring bugs unit tests miss — "the function works in isolation, but when wired to the DB it breaks."
Owner: developers + QA engineers. Tools: same unit-test frameworks but with test databases; Testcontainers (free) for spinning up real DBs in tests.

3.4 API testing (also called contract testing)

Tests API endpoints — HTTP requests in, HTTP responses out. Verifies the API contract (what fields, what status codes, what auth) is honored. Doesn't care about the UI.
Owner: QA engineers + backend developers. Tools: Postman + Newman (industry-standard, discussed at length below), REST Assured (Java), Karate (free), Insomnia, supertest (Node.js), HTTPie.

3.5 End-to-End (E2E) testing

Tests a complete user flow in a real browser — open the site, log in, click here, fill form, submit, see expected result. Slowest type. Catches integration bugs across the full stack that nothing else catches.
Owner: QA engineers (often dedicated automation engineers / SDETs). Tools: Playwright (industry-leading 2026; Microsoft-made; free), Cypress (free → $75/user/mo for team plan), Selenium (free; the original; older API), WebdriverIO (free), TestCafe (free).

3.6 Smoke testing

Quick sanity check: "Is the system minimally functional?" A small subset of critical tests (5–20) that run on every deploy. If smoke is red, no further tests run. Origin of the name: hardware engineers used to power up new circuits and see if smoke came out — smoke = fatal failure. Same idea here.
Owner: QA + DevOps (Priya runs them in our CI). Tools: subset of Playwright + Postman tests.

3.7 Sanity testing

Sometimes used interchangeably with smoke. Strict definition: narrow, deep testing on a specific area after a small change. "We just changed the fee-calculation logic — sanity-test the fee module specifically." Smoke is broad-shallow; sanity is narrow-deep.

3.8 Regression testing

Re-running existing tests to catch regressions — old features that stopped working because of a new change. Regression suites grow over time; every fixed bug becomes a regression test. The full regression suite often takes hours and runs nightly.
Owner: QA. Tools: all the above; the suite IS the test set.

3.9 User Acceptance Testing (UAT)

Real users (school admins, parents) or business stakeholders try the feature in a staging environment and confirm it works for their actual business need. Usually manual. The final human gate before release.
Owner: Product (Isha + Mohit) coordinates; QA designs test scripts. Tools: usually a test-management tool like TestRail or Xray to track UAT scripts and results; or just a structured checklist.

3.10 Alpha and Beta testing

Alpha: internal testing before public release (Schoodemy team uses the feature first). Beta: limited external testing with selected real users (a few schools try a new feature before the full rollout). Both are about catching issues that only show up in real use.

3.11 Exploratory testing

A skilled tester sits with the system and tries to break it without a script. Designed for finding what scripted tests don't anticipate. High signal, low cost, requires experienced testers. Usually 1–2 days per release, before UAT.
Owner: senior QA. Tools: notes + screen recording (Loom, etc.).

4. Non-functional testing

Functional asks 'does it work?'. Non-functional asks 'how well?'. This is where most production incidents come from: features work, but at scale or under attack they fail.

4.1 Performance testing (umbrella term)

All the ways of asking 'how does the system behave under load?'. Several sub-types:

Sub-type	What it does	Example question
Load testing	Realistic expected load. Does it cope?	Can the API handle 500 concurrent users at peak hour?
Stress testing	Push past expected load. Where does it break?	At what concurrency does the API start failing?
Spike testing	Sudden burst of traffic.	Fee deadline at 11:59 PM — 10x normal traffic for 10 minutes. Survives?
Soak / endurance testing	Sustained load over a long period.	Run at 70% load for 12 hours. Memory leak? Resource exhaustion?
Volume testing	Large data sets.	Does the report system handle a 50,000-row export?
Scalability testing	Does it scale up cleanly when we add servers?	Doubling servers should roughly double throughput.

Owner: dedicated performance engineer at scale; at our scale Priya + @Rohan run these. Tools and rough 2026 pricing:

Tool	License	Approximate cost
k6 (Grafana)	Open source (free) + cloud service (paid)	Free for self-hosted; Cloud $99–$2,499/mo
Apache JMeter	Free, open source	Free (self-hosted only)

Gatling	Open-source (free) + Enterprise	Free; Enterprise from ~$10K/year
Locust	Free, open source	Free
BlazeMeter	Cloud SaaS	$99–$999+/mo depending on virtual users
LoadRunner (OpenText)	Enterprise commercial	Tens of thousands USD / year; legacy enterprise
Artillery	Open-source + paid cloud	Free for self-hosted; Cloud from $79/mo

My recommendation for Schoodemy: k6 (free, modern, Grafana-backed, scripts in JavaScript). Priya runs it from CI before major releases.

4.2 Security testing

Whole sub-discipline. Major flavors:

Type	Acronym	What it does
Static Application Security Testing	SAST	Scans source code for vulnerabilities without running it. Finds SQL injection patterns, hardcoded credentials, unsafe deserialization, etc.
Dynamic Application Security Testing	DAST	Tests the running application from the outside, like an attacker would. Probes endpoints, fuzzes inputs.
Interactive Application Security Testing	IAST	Hybrid — instrument the running app and watch its internals while it's exercised.
Software Composition Analysis	SCA	Scans third-party dependencies (NuGet packages, npm modules) for known vulnerabilities.
Penetration testing	Pen test	Skilled humans simulate an attack on the app. Usually external firm, scheduled. The deepest form.
Vulnerability scanning	—	Automated scan of infrastructure (servers, network, ports) for known weaknesses.
Auth / authz testing	—	Specifically tests login, permissions, role boundaries, multi-tenant isolation.
Threat modeling	—	Not a test per se — an analysis. Identifies what could go wrong before code is written.

Owner: Karthik. Tools and approximate costs:

Category	Tool	Pricing (2026 approx)
SAST	SonarQube	Community free; Developer Edition $150/year/instance; Enterprise from $20K/year
SAST	Snyk	Free for small open-source; Team from $25/dev/mo; Enterprise custom
SAST	Semgrep	Free CE; Cloud Pro $40/mo/dev; Enterprise custom

SAST	GitHub Advanced Security	$21/user/mo (add-on to GitHub Enterprise)
SAST	Veracode	Enterprise commercial; tens of thousands USD/year
SAST	Checkmarx	Enterprise; tens of thousands USD/year
DAST	OWASP ZAP	Free, open source. Industry baseline.
DAST	Burp Suite	Community free; Pro $475/year/user; Enterprise from $6K/year
DAST	Acunetix	From $4,500/year
SCA	Dependabot (GitHub)	Free for public repos; included in GitHub
SCA	Snyk Open Source	Free tier; Team from $25/dev/mo
Pen test	External firms	$5K–$50K per engagement; quarterly or annual cadence common
Vuln scan	Tenable Nessus	~$3,500/year for Pro
Vuln scan	Qualys	Enterprise custom pricing

My recommendation for Schoodemy: OWASP ZAP + Semgrep (both free) as the baseline; Snyk team plan for dependency scanning (~$25/dev/mo); external pen test annually once we're shipping the new Integration Platform.

4.3 Usability testing

Real users use the product and we watch. Asks: is it intuitive? Confusing? Frustrating? Different from UAT — UAT asks 'does it do what business needs?'; usability asks 'is it pleasant to use?'.
Owner: UX (Sanya) coordinates; usability researcher leads sessions. Tools: Maze ($99–$399/mo), UserTesting ($20K+/year enterprise), Lookback ($25–$179/seat/mo), or just Zoom + Loom + a script (cheap and effective).

4.4 Accessibility testing

Verifies the product works for users with disabilities. The standard is WCAG (Web Content Accessibility Guidelines), currently version 2.2 (2026). Compliance levels: A, AA, AAA. Most enterprise contracts require AA minimum. School products often have legal accessibility obligations.
Owner: Sanya gates at design; QA verifies via automation + manual audit. Tools: axe-core (free), Pa11y (free), Lighthouse (free, Google), Accessibility Insights (free, Microsoft), WAVE (free), and manual testing with screen readers (NVDA — free; JAWS — $1,000/license; VoiceOver — built into macOS/iOS).

4.5 Compatibility testing

Does the product work across browsers, operating systems, devices, screen sizes? Critical for consumer-facing apps like e-Care that run on parent phones (Android + iOS) and desktops (Chrome / Safari / Firefox / Edge).
Owner: QA + frontend developers. Tools: BrowserStack ($29–$199+/mo), Sauce Labs ($39+/mo), LambdaTest ($15–$199/mo). Or test locally on a few devices + use cross-browser cloud only for the long tail.

4.6 Localization (l10n) and internationalization (i18n) testing

Verifies the product works in different languages, regions, currencies, date formats. Hindi + English minimum for Schoodemy; possibly regional Indian languages depending on school region.
Owner: QA. Mostly manual; some automation possible for layout / truncation issues.

4.7 Recovery / disaster testing

Verifies the system recovers from failures — database crash, network outage, server kill. Tests backup restore, failover, replication.
Owner: DevOps / SRE (Priya). Tools: usually scripted simulations + monitoring.

4.8 Chaos engineering

A modern discipline: intentionally inject failures in production-like environments to verify the system handles them gracefully. "Kill a random server every hour; does anything page?". Originated at Netflix.
Owner: SRE. Tools: Chaos Mesh (free), Gremlin (commercial — $-$$$), Litmus (free), Chaos Monkey (free, classic). Worth introducing once Schoodemy is at scale where individual server failures matter.

4.9 Compliance testing

Verifies regulatory compliance — DPDPA 2023, RBI rules for payments, POSH, RTE. Often a checklist + audit. Karthik owns this; specialized auditors for formal compliance certifications.

5. Test methodology dimensions

5.1 Manual vs automated

Manual: human runs the test. Slow per run, but flexible — good for exploratory, UAT, usability.
Automated: code runs the test. Fast, repeatable. Good for regression, performance, anything that runs often.
Rule of thumb: any test that runs more than 3–5 times should be automated.

5.2 Black-box vs white-box vs gray-box

Black-box: tester knows nothing about the internal code; tests purely through inputs / outputs. UAT, most QA testing.
White-box: tester sees and uses internal code structure. Unit testing, code coverage analysis.
Gray-box: hybrid. Tester knows some internals (e.g., DB schema) but tests through external interfaces. Most integration / API testing falls here.

5.3 Static vs dynamic

Static: analyze without running. Code reviews, linting, SAST.
Dynamic: analyze while running. Unit / integration / E2E / DAST.

5.4 Positive vs negative

Positive: "happy path" — what should work, does work.
Negative: "unhappy path" — invalid inputs, edge cases, error conditions. Most production bugs come from missed negative tests.

6. Test design techniques (how to write good tests)

Technique	What it does
Equivalence partitioning	Group inputs into classes that should behave the same. Test one representative from each.
Boundary value analysis	Test at the edges — 0, max, max-1, max+1. Most bugs live at boundaries.
Decision tables	Map combinations of inputs to expected outputs. Catches missed combinations.
State transition testing	For state-machine features (e.g., order: pending → paid → fulfilled). Test every transition.
Use case testing	Walk through a real user scenario end-to-end.
Risk-based testing	Spend more test effort on high-risk areas (payment, auth, compliance) and less on low-risk.
Pairwise testing	For features with many parameters: test pairs of combinations, not every combination (combinatorial explosion).
Property-based testing	Instead of testing specific inputs, define properties that should always hold; generate random inputs to find violations. Tools: FsCheck (.NET), fast-check (JS).

7. When tests run — the dev cycle

A modern test pipeline runs at multiple points:

When	What runs	Time budget
Developer's local machine, pre-commit	Unit + component tests for the changed code	Seconds
CI on push (Continuous Integration)	Full unit + integration + linting	~2–5 minutes; gate before merge
CI on merge request	All of above + API + smoke E2E + SAST	~10–15 minutes; gate before merge
Nightly	Full regression + full E2E + load tests on staging	~1–2 hours
Pre-deploy to staging	Subset of E2E	Few minutes
Staging (manual UAT)	UAT + exploratory	Hours / days, per release
Pre-deploy to prod	Smoke tests on staging	Few minutes
Post-deploy in prod	Synthetic monitoring + smoke	Continuous

"Shift left" — moving testing earlier in the cycle — is a constant goal. "Shift right" — monitoring in production — is the modern complement. Both matter.

8. Roles in a QA function

Role	What they do	Salary range (India 2026, approximate)

Manual QA / Test Engineer	Executes manual test cases, files bugs, runs UAT.	₹4–10 LPA (entry to mid)
Automation Engineer / SDET I	Writes automated test scripts (E2E, API). Maintains test infra.	₹8–18 LPA (entry to mid)
Senior SDET / SDET II	Designs test frameworks. Mentors junior. Owns test architecture for an app.	₹18–35 LPA
QA Lead / Test Manager	Owns test strategy across multiple apps. Coordinates with dev + product. Hiring.	₹25–50 LPA
Performance Engineer	Specializes in load / stress / capacity. Often hybrid SRE.	₹20–40 LPA
Security Tester / Pen Tester	Specializes in security testing. OSCP / OSCE certs common.	₹15–35 LPA; pen testers often consultants charging ₹50K–₹5L per engagement
Test Architect / Principal SDET	Cross-org test strategy. Tooling decisions. Patterns.	₹40–80 LPA
QA Director / VP	Multi-team org-level QA leadership.	₹60L+ LPA

LPA = Lakhs Per Annum (Indian salary convention; 1 lakh = 100,000 rupees).

9. The full industry tools landscape — with 2026 pricing

All prices approximate, per-user-per-month unless noted, in USD. Indian pricing often different and sometimes negotiable.

9.1 Unit testing frameworks (all free)

Tool	Language	Notes
xUnit	.NET	Industry standard for .NET (Microsoft-backed in spirit, community-maintained)
NUnit	.NET	Older, still widely used
MSTest	.NET	Microsoft's built-in; often used in Visual Studio shops
JUnit	Java	The original; industry standard for Java
TestNG	Java	Java alternative with more features
Jest	JS/TS	Created by Facebook; React default; widely used
Vitest	JS/TS	Modern replacement for Jest; Vite-native; growing fast
Mocha + Chai	JS	Classic Node.js stack
pytest	Python	Industry standard for Python
RSpec	Ruby	Industry standard for Ruby
Go test	Go	Built into the language

9.2 API testing

Tool	License	Cost (2026 approx)
Postman	Freemium SaaS	Free for individuals; Basic $14/user/mo; Pro $29/user/mo; Enterprise from $49/user/mo

Newman	Open source CLI	Free; companion to Postman
Insomnia	Freemium	Free + Team plan from $5/user/mo
REST Assured	Open source	Free; Java-based
Karate	Open source	Free; behavior-driven syntax
Bruno	Open source	Free; Postman alternative, git-friendly
Hoppscotch	Open source	Free; web-based
SoapUI	Freemium	Free + Pro from $819/year (legacy, mostly SOAP)

9.3 E2E and UI testing

Tool	License	Cost (2026 approx)
Playwright	Open source (Microsoft)	Free. Industry-leading 2026.
Cypress	Freemium	Free for OSS / small; Team $75/user/mo; Business $300/user/mo
Selenium	Open source	Free. The original; older API.
WebdriverIO	Open source	Free. Selenium-compatible.
TestCafe	Open source	Free.
Puppeteer	Open source	Free; lower-level browser control.
Nightwatch.js	Open source	Free.
Robot Framework	Open source	Free; keyword-driven syntax.

9.4 Mobile testing

Tool	Platform	Cost (2026 approx)
Appium	Cross-platform	Free, open source. Industry standard.
Espresso	Android	Free; Google's native framework.
XCUITest	iOS	Free; Apple's native framework.
Detox	React Native	Free.
Maestro	Mobile	Free + Cloud paid; modern.
BrowserStack	Cloud device farm	$29 individual / $199+ team
Sauce Labs	Cloud device farm	$39+/mo
LambdaTest	Cloud device farm	$15–$199/mo
Firebase Test Lab	Cloud device farm	Free tier + pay per device-hour

9.5 Visual regression testing

Detects unintended visual changes — "the button moved 3 pixels and we didn't mean to."

Tool	License	Cost (2026 approx)
Percy (BrowserStack)	SaaS	$199–$699/mo
Chromatic (Storybook)	SaaS	$149–$649/mo
Applitools Eyes	SaaS, AI-powered	$300+/mo
BackstopJS	Open source	Free.
Playwright snapshots	Open source	Free; built into Playwright.

9.6 Test management (organizing test cases + runs)

Tool	Cost (2026 approx)
TestRail	$35–$70/user/mo

Xray (Jira plugin)	$1.25–$5/user/mo (additional to Jira)
Zephyr Scale	$10–$15/user/mo
qTest	Enterprise custom
PractiTest	$49–$79/user/mo
TestLodge	$24–$249/mo flat
Notion / Linear (DIY)	Free–$10/user/mo; works for small teams

9.7 Bug / defect tracking

Tool	Cost (2026 approx)
Jira (Atlassian)	~$8.15/user/mo Standard; $16+ Premium
Linear	$8/user/mo Standard
GitHub Issues	Free with GitHub
GitLab Issues	Free with GitLab
Asana	$10.99–$24.99/user/mo
Bugzilla	Free, open source
Mantis	Free, open source
ClickUp	$7–$19/user/mo

9.8 Test reporting / dashboards

Tool	Cost (2026 approx)
Allure	Free, open source
ReportPortal	Free, open source
Newman HTML Extra reporter	Free
Playwright HTML reporter	Free, built-in
Cypress Cloud / Dashboard	Free up to 500 results; from $75/user/mo
Currents.dev	Cypress + Playwright reporter; from $99/mo

9.9 CI/CD (where automated tests run)

Tool	Cost (2026 approx)
GitLab CI/CD	Free up to 400 CI minutes/mo; Premium $29/user/mo; Ultimate $99/user/mo
GitHub Actions	Free for public repos; 2,000 mins/mo on Free private; pay per minute beyond
Jenkins	Free, open source. Self-hosted. Industry workhorse.
CircleCI	Free 6,000 build mins/mo; Performance $15/user/mo
Azure Pipelines	$40/parallel job/mo Microsoft-hosted; or self-host free
Bitbucket Pipelines	Free 50 build mins; Standard $3/user/mo
TeamCity (JetBrains)	Free Professional (limited); Enterprise from ~$2K/year
Buildkite	$15+/user/mo, self-hosted agents

9.10 Test data + mocking

Tool	Purpose	Cost
Faker (libraries in many languages)	Generate fake data	Free
Mockaroo	Browser-based fake data generator	Free + Pro from $5/mo
Tonic.ai	Production-like synthetic data	Enterprise custom
GenRocket	Enterprise test data generation	Enterprise custom

WireMock	API mocking	Free
MockServer	API mocking	Free
Mock Service Worker (MSW)	Browser-side API mocking	Free
Mountebank	Cross-protocol mocking	Free

9.11 Performance / load (already listed in §4.1)

See §4.1 above. Schoodemy default: k6 (free).

9.12 Security (already listed in §4.2)

See §4.2 above. Schoodemy default: OWASP ZAP + Semgrep + Snyk team plan.

9.13 Observability (production monitoring, "shift right")

Tool	License	Cost (2026 approx)
Datadog	SaaS	$15–$23/host/mo Infrastructure; more for APM / logs
New Relic	SaaS	$25–$99/user/mo Standard; Free tier 100GB ingest
Grafana stack (Loki, Prometheus, Tempo)	Open source + Cloud	Free self-hosted; Cloud from $8/mo
Sentry	SaaS + open source	Free; Team $26/mo+; Business $80/mo+
Dynatrace	Enterprise SaaS	Tens of thousands USD/year
Honeycomb	SaaS	$0–$130/user/mo
Splunk	Enterprise	$$$+; volume-based
Elastic / OpenSearch	Open source + SaaS	Free self-hosted; Cloud $16+/mo

10. How to build a QA department from scratch

My opinionated path, drawing on what I've seen work:

10.1 Phase 0 — strategy + tooling decisions (weeks 1–4)

- Decide test-pyramid targets — what % unit, integration, E2E? Typical: 70 / 20 / 10.
- Pick the toolset — unit framework per language, one API tool, one E2E tool, one bug tracker. Don't overcomplicate. Costs minimal at this stage.
- Establish gating criteria — what tests must pass for what stages (R0, R13 in our case).
- Set up CI integration foundations.

10.2 Phase 1 — first automation engineer + framework (months 2–4)

- Hire 1 senior SDET / automation engineer.
- Build the first E2E framework on one app. Pilot.
- Run the first regression suite.
- Establish bug-triage cadence.
- Budget: ~₹20–35 LPA for the SDET + tooling cost minimal (most free).

10.3 Phase 2 — expand coverage + add manual QA (months 4–8)

- Hire 1–2 manual QA testers to handle UAT + exploratory + regression triage.
- Roll out the automation pattern to more apps.
- Establish performance testing capability.

- Budget: ~₹15–25 LPA for manual testers; tools still mostly free; some cloud device farm cost (BrowserStack ~$199/mo).

10.4 Phase 3 — specialization + scale (year 2+)

- Add a dedicated performance engineer if scale demands.
- Bring security testing in-house or contract pen tests annually.
- Establish test data management discipline.
- Consider visual regression + accessibility automation tools.
- Promote first SDET to lead; hire mid-level SDETs underneath.

10.5 Org structure to aim for (at our 27-app scale)

By the time Schoodemy has 27 apps in production, a healthy QA function looks like:

Role	Count	Reports to
QA / Test Lead (Mira)	1	Mohit (per v4.2)
Senior SDETs / Test Architects	2–3	Mira
Automation Engineers (mid-level SDETs)	4–6	Senior SDETs
Manual QA / Test Engineers	3–5	Mira / Senior SDETs
Performance Engineer	1	Mira (hybrid with Priya's SRE)
Security Tester / hired pen test consultants	0–1 + external	Karthik on technical, Mira on test ops

Total full-time: ~11–16 people for a 27-app shop. Today @Tara is the entire automation team; we have hiring to do.

11. Metrics that matter (what to track)

Metric	What it tells you
Test coverage %	How much of the codebase is exercised by tests. Aim 70–80% for critical apps; 100% is a smell (you're testing trivial code).
Pass rate	Of tests that ran, how many passed. Should be >95%; below that, flakiness is killing trust.
Defect escape rate	Bugs found in production / total bugs. Lower is better. Industry typical: ~5–15%.
Mean time to detect (MTTD)	How long between a bug being introduced and being detected. Lower is better.
Mean time to resolve (MTTR)	How long to fix a found bug. Lower is better.
Test execution time	How long the full suite takes. <30 min is healthy; >2 hours is a problem.
Flaky test rate	Tests that pass / fail nondeterministically. Above 1% is bad — trust collapses fast.
Test maintenance cost	Time spent fixing tests vs writing new ones. Should be <30%.
Customer-reported bugs / week	Bugs found by users post-release. Trending down is the goal.

12. Specific to Schoodemy — applying all this

Mapping this universe to Schoodemy's v4.2 work flow:

Stage in our work flow	Test type at that stage
Stage 0 (Product / BA Gatekeeping)	Acceptance criteria defined; nothing tested yet.
Stage 1 (UX Design)	Accessibility review at design time. Sanya.
Stage 2 (Architecture)	Threat modeling. Karthik.
Stages 3–6 (Build)	Developers write unit + component tests as they build. R0 expects coverage.
Stage 7 (Security)	SAST scan; auth/authz tests; DPDPA compliance check.
Stage 8 (Test)	Mira + @Tara design test plan; write/run API + E2E tests; regression.
Stage 9 (R0 Audit)	Parallel domain audits including QA. Hard gate.
Stage 10 (DevOps)	Smoke tests on staging deploy; load tests pre-release.
Stage 11 (Data / BI)	Devika tests warehouse loads.
Stage 12 (Docs)	Riya verifies docs by walkthrough.
Stage 13 (Ship)	Final smoke on prod; synthetic monitoring active.

12.1 Recommended Schoodemy test stack — final view

Layer	Tool	Approximate annual cost (2026)
Unit (.NET)	xUnit	Free
Unit (Angular)	Vitest	Free
API	Postman + Newman	Postman Team ~$14–29/user/mo (~$3–8K/year for ~20 users)
E2E	Playwright	Free
Mobile E2E	Appium + cloud device farm (BrowserStack ~$199/mo)	~$2.4K/year
Performance	k6 (self-hosted)	Free
Security SAST	Semgrep + Snyk team	~$25/dev/mo × ~15 devs = ~$4.5K/year
Security DAST	OWASP ZAP	Free
Pen test	External annually	~$10K–$25K per engagement
Accessibility	axe-core + manual screen-reader checks	Free (NVDA) + ~$1K (JAWS license if needed)
CI/CD	GitLab CI/CD (Premium tier)	~$29/user/mo × ~30 users = ~$10K/year
Test management	Linear or Notion (no separate tool)	Already paid for
Bug tracking	GitLab Issues	Included in GitLab
Monitoring	Grafana stack (self-hosted)	Free + infra cost
Visual regression	Playwright snapshots (built-in)	Free

Total tool cost: roughly $25–35K USD / year (~₹20–30 lakh) — modest, since most are free / open source. Most cost is in headcount, not tools. That's normal.

13. Common QA-shop anti-patterns to avoid

- "QA tests at the end" — bugs found in week 6 cost 100x more than week 1. Test continuously.
- "Automate everything" — exploratory + UAT need humans; don't try to automate them.
- "100% test coverage" — diminishing returns past 80%; you'll be testing trivial code.
- "Flaky tests are okay if we re-run" — flakiness erodes trust; fix or delete them.
- "Just hire more manual QA" — manual QA doesn't scale; automate the repeatable.
- "QA owns quality" — quality is everyone's job; QA enables it but devs own their code.

- "Test scripts are documentation" — tests are not docs; write actual docs.
- "Don't test in production" — actually you must (smoke + synthetic monitoring); just don't test destructively.

14. Glossary

Term	Definition
SDET	Software Development Engineer in Test. Engineer who writes test automation. More technical than manual QA.
QA	Quality Assurance. The broader discipline.
UAT	User Acceptance Testing. Business stakeholders verify the feature meets their need.
CI/CD	Continuous Integration / Continuous Delivery (or Deployment). The automated pipeline that runs tests + ships code.
SAST	Static Application Security Testing. Code-level security scanning.
DAST	Dynamic Application Security Testing. Running-app security testing.
IAST	Interactive Application Security Testing. Hybrid SAST + DAST.
SCA	Software Composition Analysis. Third-party dependency vulnerability scanning.
WCAG	Web Content Accessibility Guidelines. Accessibility standard.
MTTD	Mean Time To Detect. Average time from bug introduced to bug found.
MTTR	Mean Time To Resolve. Average time from bug found to bug fixed.
SLA	Service Level Agreement. Commitment to a level of service.
SLO	Service Level Objective. Internal target.
Flaky test	A test that passes / fails inconsistently without code changes.
Mock	A fake version of a dependency used in testing.
Stub	A simpler form of mock — just returns canned values.
Fixture	Setup data / state for a test.
Test double	Generic term for mock / stub / fake.
Shift left	Move testing earlier in the dev cycle.
Shift right	Add monitoring + testing in production.
Coverage	% of code exercised by tests.
Regression	A feature that used to work now doesn't.
Smoke	Quick sanity test that the system is up.
Edge case	Unusual / boundary input. Where most bugs live.
Happy path	The expected, successful flow.
Sad path / unhappy path	Error / exception flows.
Pyramid (test pyramid)	Mike Cohn's model: lots of unit tests, fewer integration, fewest E2E.
Trophy (testing trophy)	Kent C. Dodds' counter-model: heavier on integration; lighter on unit + E2E.

BDD	Behavior-Driven Development. Tests written in human-readable scenario format (Given / When / Then).
TDD	Test-Driven Development. Write the test first, then the code to make it pass.
Gherkin	The Given/When/Then syntax used in BDD tools like Cucumber.
CI	Continuous Integration.
CD	Continuous Delivery / Deployment.
DAST	Dynamic Application Security Testing.
Pen test	Penetration test — authorized simulated attack on a system.
Red team	A team simulating attackers in security testing.
Blue team	A team defending against red team attacks.
Chaos engineering	Intentionally inject failures to test resilience.
Canary release	Deploy to a small subset of users first; watch; then roll out.
Blue-green deploy	Run old + new versions simultaneously; switch over atomically.
Feature flag	Code switch that lets you turn a feature on/off without deploying.
Mutation testing	Mutate the code; check if tests catch the change. Tests the tests.

15. Where to read further

- "Lessons Learned in Software Testing" by Kaner / Bach / Pettichord — old but foundational.
- "Continuous Delivery" by Jez Humble / David Farley — testing in CI/CD.
- Martin Fowler's blog (martinfowler.com) — testing articles, especially "TestPyramid".
- Kent C. Dodds' "Testing Trophy" articles — modern counter-take on the pyramid.
- Google Testing Blog (testing.googleblog.com) — testing at scale.
- OWASP Testing Guide — security testing reference.
- WCAG 2.2 spec — accessibility.
- For us specifically: revisit this doc + the v4.2 standards (in /Tech Department/standards/).