

Welcome to QA/Software Testing

- QA is risk management for product quality
- Testing is a team sport across SDLC
- You will learn manual + automation + quality engineering

References: <https://www.istqb.org/certifications/certified-tester-foundation-level> | <https://agilemanifesto.org/>

How to Use This Course

- Watch slides first, then practice labs
- Use scripts as your narration guide
- Build a portfolio project by chapter 12

References: <https://www.istqb.org/certifications/certified-tester-foundation-level>

Evergreen + Trending Topics Covered

- Evergreen: test design, risk, defects, automation strategy
- Trending: shift-left, CI/CD quality gates, observability
- Modern tooling: Playwright, API/contract testing, k6

References: <https://dora.dev/> | <https://playwright.dev/docs/intro>

Learning Outcomes

- Design effective tests with clear coverage
- Build maintainable UI/API automated checks
- Integrate quality gates into CI/CD pipelines

References: <https://docs.github.com/en/actions> | <https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment>

What is Testing?

- Testing evaluates software quality and risk
- Testing provides information, not absolute proof
- Quality includes functional and non-functional attributes

References: <https://www.istqb.org/certifications/certified-tester-foundation-level> | <https://iso25000.com/index.php/en/iso-25000-standards/iso-25010>

Quality Attributes (ISO 25010)

- Key attributes include performance, security, maintainability
- Functional correctness is only one part of quality
- Test strategy should map to target attributes

References: <https://iso25000.com/index.php/en/iso-25000-standards/iso-25010>

Testing Principles (ISTQB-oriented)

- Early testing saves cost and rework
- Defects cluster in a small number of modules
- Tests wear out and need regular review

References: <https://www.istqb.org/certifications/certified-tester-foundation-level>

Static vs Dynamic Testing

- Static: reviews, walkthroughs, linters, SAST
- Dynamic: executing software with test cases
- Both are complementary and should coexist

References: <https://www.istqb.org/certifications/certified-tester-foundation-level> | <https://owasp.org/www-project-top-ten/>

Test Levels and Types

- Levels: component, integration, system, acceptance
- Types: functional, non-functional, structural
- Select level/type based on risk and feedback speed

References: <https://martinfowler.com/articles/practical-test-pyramid.html> | <https://www.selenium.dev/documentation/>

Designing Useful Test Cases

- Start from requirements and acceptance criteria
- Include positive, negative, and edge scenarios
- Keep tests independent and reproducible

References: <https://www.istqb.org/certifications/certified-tester-foundation-level>

Equivalence Partitioning

- Group inputs into expected-behavior classes
- Test one representative per valid/invalid class
- Reduces case count while preserving coverage

References: <https://www.istqb.org/certifications/certified-tester-foundation-level>

Boundary Value Analysis

- Defects often occur near input boundaries
- Test min, max, and just-inside/outside values
- Apply to ranges, lengths, dates, and limits

References: <https://www.istqb.org/certifications/certified-tester-foundation-level>

Decision Table Testing

- Use rules for condition/action combinations
- Great for pricing, eligibility, permissions
- Ensures business-rule completeness

References: <https://www.istqb.org/certifications/certified-tester-foundation-level>

State Transition Testing

- Model states and allowed transitions
- Test valid and invalid state changes
- Useful for workflows and lifecycle systems

References: <https://www.istqb.org/certifications/certified-tester-foundation-level>

Web Testing Essentials

- Verify UX-critical paths and core flows
- Check cross-browser and responsive behavior
- Include accessibility and error-state checks

References: <https://www.selenium.dev/documentation/> | <https://web.dev/learn/accessibility>

API Testing Fundamentals

- Validate status, schema, and business rules
- Check idempotency, auth, and error contracts
- Automate API checks for fast feedback

References: <https://learning.postman.com/docs/tests-and-scripts/write-scripts/test-scripts/>

Contract Testing (CDC)

- Verify producer-consumer API contracts
- Catch integration drift earlier
- Reduce dependence on brittle full E2E

References: <https://pact.io/> | <https://docs.pact.io>

Exploratory Testing for Web/API

- Use charters to focus time-boxed exploration
- Document observations and new risks quickly
- Feed findings back into regression suites

References: <https://www.istqb.org/certifications/certified-tester-foundation-level>

Security and Abuse Cases

- Include auth, authorization, injection, and session tests
- Map critical checks to OWASP Top 10
- Prioritize by business impact and exposure

References: <https://owasp.org/www-project-top-ten/>

Automation Strategy First

- Automate repeatable, high-value, stable checks
- Keep manual testing for discovery and UX judgment
- Treat automation as engineering product code

References: <https://martinfowler.com/articles/practical-test-pyramid.html>

Test Pyramid in Practice

- More fast unit/service checks, fewer UI E2E
- UI tests cover user-critical journeys only
- Balance confidence with execution cost

References: <https://martinfowler.com/articles/practical-test-pyramid.html>

Flaky Test Mitigation

- Quarantine non-deterministic tests quickly
- Remove shared state and timing dependencies
- Track reliability trends over time

References: <https://testing.googleblog.com/2016/05/flaky-tests-at-google-and-how-we.html>

Maintainable Test Architecture

- Use page objects/screenplay where appropriate
- Centralize fixtures and test data builders
- Keep assertions explicit and readable

References: <https://playwright.dev/docs/intro> | <https://docs.pytest.org/en/stable/>

BDD: When and When Not

- Use Gherkin for shared understanding
- Avoid turning feature files into low-level scripts
- Prefer business-readable scenarios

References: <https://cucumber.io/docs/gherkin/reference/>

Why Playwright + Python

- Cross-browser support with modern primitives
- Strong locator model and web-first assertions
- Python ecosystem integrates well with QA workflows

References: <https://playwright.dev/docs/intro> | <https://docs.pytest.org/en/stable/>

Core Playwright Workflow

- Launch browser/context/page deterministically
- Use resilient locators and explicit assertions
- Collect traces/screenshots on failures

References: <https://playwright.dev/docs/intro> | <https://playwright.dev/docs/test-assertions>

Pytest Integration Patterns

- Use fixtures for setup/teardown reuse
- Parametrize tests to scale coverage
- Use markers to target suites by risk

References: <https://docs.pytest.org/en/stable/>

Data-Driven and Environment-Aware Tests

- Read config from environment variables
- Separate secrets from source code
- Use clean test data lifecycle per run

References: <https://12factor.net/config>

Debugging Failing UI Tests

- Reproduce with headed mode and traces
- Inspect network/console before changing assertions
- Fix product bugs before patching tests

References: <https://playwright.dev/docs/trace-viewer-intro>

Performance Testing Basics

- Define SLO-aligned performance goals
- Run baseline, load, stress, and soak tests
- Analyze response time and error-rate behavior

References: <https://grafana.com/docs/k6/latest/> | <https://sre.google/sre-book/service-level-objectives/>

k6 for Practical Performance Checks

- Write JavaScript-based load scenarios
- Integrate runs into CI for regressions
- Use thresholds to fail builds on critical degradation

References: <https://grafana.com/docs/k6/latest/>

Accessibility Testing Foundations

- Use WCAG as baseline standard
- Automate common checks with tools like axe
- Add keyboard and screen-reader manual checks

References: <https://www.w3.org/TR/WCAG22/> | <https://www.deque.com/axe/>

Security Testing in QA

- Prioritize top web risks from OWASP
- Include misuse/abuse scenarios in test design
- Collaborate with AppSec for deeper assessments

References: <https://owasp.org/www-project-top-ten/>

Reliability and Observability

- Use logs, metrics, traces to investigate failures
- Design tests to expose operational risk
- Collaborate with SRE/DevOps on incident learnings

References: <https://opentelemetry.io/docs/> | <https://sre.google/sre-book/table-of-contents/>

CI/CD and Quality Gates

- Run automated tests on every significant change
- Block merges on critical quality failures
- Keep pipelines fast to preserve developer flow

References: <https://docs.github.com/en/actions> | <https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment>

GitHub Actions for QA

- Define workflows as code in YAML
- Split jobs by test type and parallelize
- Publish reports/artifacts for visibility

References: <https://docs.github.com/en/actions>

Branching and Integration Strategy

- Prefer short-lived branches and frequent merges
- Use trunk-based practices to reduce merge risk
- Keep main branch releasable

References: <https://trunkbaseddevelopment.com/>

Release Confidence and Versioning

- Use semantic versioning for clear change signals
- Tie release decisions to evidence, not intuition
- Track rollback readiness and known risks

References: <https://semver.org/>

Shift-Left and Shift-Right Together

- Shift-left: prevent defects earlier
- Shift-right: monitor and validate in production
- Use both for complete quality feedback loops

References: <https://dora.dev/> | <https://opentelemetry.io/docs/>

Defect Reporting That Helps Teams

- Write clear repro steps and expected vs actual
- Attach evidence: logs, screenshots, environment
- Prioritize by user/business impact

References: <https://www.istqb.org/certifications/certified-tester-foundation-level>

QA Metrics That Matter

- Track leading indicators (flake rate, escaped defects)
- Track delivery outcomes alongside quality metrics
- Use metrics to improve systems, not punish people

References: <https://dora.dev/> | <https://testing.googleblog.com/2016/05/flaky-tests-at-google-and-how-we.html>

Career Roadmap: QA to Quality Engineer

- Strengthen testing fundamentals and product thinking
- Build coding skills for automation and tooling
- Develop CI/CD, observability, and risk communication

References: <https://docs.pytest.org/en/stable/> | <https://playwright.dev/docs/intro>

Capstone Project Blueprint

- Select a real web app with API dependencies
- Create test strategy + automated regression core
- Run in CI and publish quality dashboard artifacts

References: <https://docs.github.com/en/actions> | <https://playwright.dev/docs/intro> | <https://learning.postman.com/docs/tests-and-scripts/write-scripts/test-scripts/>

