# 100 Conflicts, Resolutions & Steps

*Mobile & Web Test Automation — with step-by-step implementation for each resolution*
*Technical, tooling, and team/process conflicts common across the industry, 2026*

## A. Mobile Automation — Technical / Tooling Conflicts

1. **Conflict:** Flaky tests from animations/transitions.
**Resolution:** Disable animations in test builds; wait on explicit conditions, not sleep().

   **Step 1:** Add a build flag that sets Android's animator-duration-scale to 0 and disables iOS UIView animations at launch for QA/test builds.
   **Step 2:** Replace fixed sleep() calls with explicit waits on a stable signal (element visible/enabled, network idle).
   **Step 3:** Have CI apply the animation-disable flag automatically before every run, not manually per test.

2. **Conflict:** Dynamic resource-ids break locators.
**Resolution:** Use accessibility-id/content-desc or testID instead of index-based XPath.

   **Step 1:** Add stable accessibility identifiers (iOS accessibilityIdentifier, Android contentDescription/testTag) to every interactive element in the app code.
   **Step 2:** Update the locator layer to query by these identifiers instead of resource-id/index/XPath.
   **Step 3:** Add a CI check that fails a PR if a new interactive component ships without an identifier.

3. **Conflict:** iOS vs Android locator strategies diverge.
**Resolution:** Abstract locators behind a page-object layer with platform-specific selectors.

   **Step 1:** Create one page-object class per screen with platform-agnostic method names.
   **Step 2:** Inside each method, branch by platform to the correct native locator.
   **Step 3:** Keep test scripts calling only the page-object API, never raw platform locators.

4. **Conflict:** Real device fragmentation (screens, OS versions).
**Resolution:** Use a device farm (BrowserStack/Sauce Labs/Firebase Test Lab) with a defined device matrix, not one physical device.

   **Step 1:** Define the device/OS matrix from real analytics (top devices and OS versions by user share).
   **Step 2:** Provision that matrix on a device farm instead of buying physical units per model.
   **Step 3:** Wire CI to fan the suite out across the farm in parallel and aggregate results into one report.

5. **Conflict:** Emulator/simulator behavior mismatches real devices (permissions, sensors, push).
**Resolution:** Run smoke tests on emulators; run full regression on real devices.

   **Step 1:** Tag tests as emulator-safe vs requires-real-device (sensors, push, biometrics, carrier features).
   **Step 2:** Run the emulator-safe subset on every PR for fast feedback.
   **Step 3:** Run the full suite, including real-device-only tests, on a device farm nightly or pre-release.

6. **Conflict:** Appium session startup is slow.
**Resolution:** Reuse driver sessions across tests; parallelize via a device-farm grid.

   **Step 1:** Group tests sharing app state into one Appium session instead of restarting per test.
   **Step 2:** Reset only the necessary state between tests within a session rather than killing the app.
   **Step 3:** Parallelize across multiple device-farm sessions to offset remaining per-session cost.

7. **Conflict:** App state (login) persists across tests.
**Resolution:** Reset app state — clear data or reinstall — before each test.

**Step 1:** Add a setup hook that clears app data (or reinstalls) before every test class/session.
**Step 2:** Seed required starting state via API call or ADB command, not UI navigation.
**Step 3:** Confirm isolation by running a test alone and inside the full suite — results must match.

8. **Conflict:** Deep link / push notification tests are flaky.
**Resolution:** Mock notification payloads via ADB intents or simulated push instead of real push infrastructure.

**Step 1:** Replace real push calls in tests with ADB broadcast (Android) or simulated payload injection (iOS simulator).
**Step 2:** Define fixture payloads for each notification type the app handles.
**Step 3:** Assert on the app's resulting UI/navigation state, not on push delivery itself.

9. **Conflict:** Biometric (Face ID/Touch ID) automation blocked by OS security.
**Resolution:** Use the simulator's biometric enrollment APIs or a test-only bypass build flag.

**Step 1:** Enroll a test fingerprint/face in the simulator via its enrollment command.
**Step 2:** Trigger a matching or non-matching biometric event programmatically from the test.
**Step 3:** For environments without simulation support, add a test-only bypass flag behind a debug menu.

10. **Conflict:** Network condition variability (3G/4G/offline).
**Resolution:** Simulate network conditions (Charles proxy, Appium network shaping) rather than testing on real carrier networks.

**Step 1:** Configure a network-conditioning proxy with named profiles (3G, offline, high-latency).
**Step 2:** Route the test device/simulator's traffic through the proxy before running network-dependent tests.
**Step 3:** Assert on the app's degraded-network behavior (retry banners, offline mode) per profile.

11. **Conflict:** Gesture automation (swipe, pinch) is inconsistent across OS versions.
**Resolution:** Use the W3C Actions API instead of the deprecated TouchAction API.

**Step 1:** Replace TouchAction-based gesture code with the W3C Actions (pointer input) API.
**Step 2:** Re-tune gesture coordinates and durations against current OS versions.
**Step 3:** Run the gesture suite across the full OS-version matrix before removing old TouchAction code.

12. **Conflict:** App version drift breaks the test suite.
**Resolution:** Pin the app build version to the test-suite version in CI; gate releases on a green suite.

**Step 1:** Tag each build artifact with a version ID stored alongside the suite's expected-compatible version.
**Step 2:** Have CI fail fast if the build under test doesn't match the pinned range.
**Step 3:** Update the pinned version deliberately during release, re-validating the suite against the new build first.

13. **Conflict:** iOS code signing/provisioning conflicts in CI.
**Resolution:** Use fastlane match for centralized signing certificates.

**Step 1:** Set up shared storage for fastlane match holding certificates and provisioning profiles.
**Step 2:** Configure every CI runner to fetch credentials via match instead of local keychains.
**Step 3:** Rotate certificates through match when they expire, rather than manual re-uploads per machine.

14. **Conflict:** Android Gradle/SDK version conflicts across CI runners.
**Resolution:** Pin Gradle wrapper and SDK versions; use containerized CI images.

**Step 1:** Commit the Gradle wrapper and a fixed compileSdk/targetSdk to the repo.
**Step 2:** Build a containerized CI image with the exact SDK/NDK/build-tools versions needed.
**Step 3:** Update pinned versions and the container image together in one reviewed change.

15. **Conflict:** Appium driver version incompatible with a new OS release.
**Resolution:** Pin Appium and driver versions; upgrade on a schedule after regression testing, not immediately on OS release.

**Step 1:** Pin the Appium server and platform driver versions explicitly in the project's dependencies.
**Step 2:** Test the upgrade path on a branch against the full suite when a new OS ships.
**Step 3:** Roll upgrades out on a schedule (e.g., monthly), not reactively on release day.

16. **Conflict:** Parallel execution causes device resource contention.
**Resolution:** Use device-farm queuing/sharding instead of a shared single-device pool.

**Step 1:** Move from a fixed local device pool to a farm with built-in queuing/sharding.
**Step 2:** Configure the runner to request N parallel slots and let the farm schedule them.
**Step 3:** Monitor queue wait times and adjust concurrency to balance cost vs speed.

17. **Conflict:** In-app webviews mix native and web automation.
**Resolution:** Explicitly switch Appium context (NATIVE_APP vs WEBVIEW) per step.

**Step 1:** Identify every screen that embeds a webview and mark those steps explicitly.
**Step 2:** Call the driver's context-switch API before web interactions and back after.
**Step 3:** Assert the expected context is active before each interaction to catch silent switch failures.

18. **Conflict:** Third-party SDK popups (ads, analytics) interrupt flows.
**Resolution:** Disable third-party SDKs in the QA build configuration.

**Step 1:** Add a build-time flag that disables ad SDKs and similar third-party UI in QA/test builds.
**Step 2:** Confirm the flag only suppresses interrupting UI, not core logic under test.
**Step 3:** Periodically test with the flag off too, to ensure production behavior hasn't silently drifted.

19. **Conflict:** Locale/language testing multiplies the test matrix.
**Resolution:** Parameterize tests by locale; run a subset per PR, full matrix nightly.

**Step 1:** Parameterize test cases by locale as one reusable definition, not copy-pasted per locale.
**Step 2:** Run a representative subset (RTL/LTR, longest strings) on every PR.
**Step 3:** Run the full locale matrix on a nightly or pre-release schedule.

20. **Conflict:** Permission dialogs block automation.
**Resolution:** Pre-grant permissions via ADB or simulator defaults before the test run.

**Step 1:** List all runtime permissions the app requests.
**Step 2:** Pre-grant them via ADB or simulator config before app launch.
**Step 3:** Keep one test exercising the real permission-prompt flow so that path stays covered.

21. **Conflict:** Battery/performance throttling causes timing flakiness.
**Resolution:** Disable low-power mode on test devices; fix explicit waits instead of masking with retries.

**Step 1:** Disable low-power/battery-saver mode on all test devices in the lab/farm config.
**Step 2:** Replace timing assumptions with explicit waits tied to real signals.
**Step 3:** Occasionally re-run under intentional throttling to confirm the app itself still behaves correctly.

22. **Conflict:** Appium results diverge from native XCTest/Espresso results.
**Resolution:** Treat native frameworks as source of truth for platform-specific logic; use Appium only for cross-platform E2E.

**Step 1:** Keep XCTest/Espresso as the authority for platform-specific components and OS integrations.
**Step 2:** Reserve Appium for cross-platform, end-to-end journeys only.
**Step 3:** When results conflict, trust the native result and investigate the Appium failure as a tooling artifact.

23. **Conflict:** Over-the-air app updates on real devices cause flakiness.
**Resolution:** Disable auto-update on the test device pool.

**Step 1:** Configure test devices/images to disable automatic OS and app-store updates.
**Step 2:** Pin the OS version per device profile explicitly rather than 'latest'.

**Step 3:** Schedule deliberate OS-version upgrades of the pool, re-validating the suite at that point.

24. **Conflict:** List/table virtualization makes elements “not visible”.
**Resolution:** Scroll to the element explicitly before interacting; don't assume it's in the DOM.

**Step 1:** Build a shared scroll-to-element helper that scrolls until the target renders.
**Step 2:** Replace direct find-and-tap calls on list items with this helper across the suite.
**Step 3:** Make the failure message distinguish 'doesn't exist' from 'exists but not scrolled into view'.

25. **Conflict:** Full device-matrix runs on every commit blow the CI budget.
**Resolution:** Tier testing — smoke on PR, full matrix nightly or pre-release.

**Step 1:** Classify tests into smoke (every PR), regression (nightly), and full-matrix (pre-release) tiers.
**Step 2:** Configure CI triggers so only the smoke tier runs on every commit.
**Step 3:** Track CI spend per tier monthly and adjust tier boundaries as needed.

## B. Web Automation — Technical / Tooling Conflicts

26. **Conflict:** Flaky tests from async JS rendering.
**Resolution:** Use an auto-waiting framework (Playwright) or explicit wait-for-selector; avoid hard sleeps.

**Step 1:** Migrate interactions/assertions to a framework with built-in auto-waiting instead of manual sleeps.
**Step 2:** Where sleeps remain in a legacy suite, replace each with an explicit wait-for-selector/response call.
**Step 3:** Add a lint rule flagging any hardcoded wait/sleep call in review.

27. **Conflict:** Cross-browser rendering differences.
**Resolution:** Run the critical-path suite across engines (Chromium, WebKit, Firefox) via one framework's multi-browser API.

**Step 1:** Identify the critical journeys (checkout, login, core feature) that must work everywhere.
**Step 2:** Run that subset against Chromium, WebKit, and Firefox in the same CI run.
**Step 3:** Keep full regression Chromium-only unless a browser-specific bug is reported, to control cost.

28. **Conflict:** Shadow DOM/web components are hard to locate.
**Resolution:** Use the framework's piercing selectors (e.g. Playwright's shadow-piercing CSS).

**Step 1:** Identify components built with Shadow DOM/web components.
**Step 2:** Use the framework's shadow-piercing selector syntax to reach elements inside shadow roots.
**Step 3:** Encapsulate shadow-aware selectors in the page-object layer so authors don't need DOM internals.

29. **Conflict:** iframe context-switching errors.
**Resolution:** Explicitly switch frame context before interacting; don't assume top-level DOM.

**Step 1:** Map which UI sections render inside iframes (payment widgets, embeds).
**Step 2:** Explicitly switch context to the target frame before interacting, and back after.
**Step 3:** Add a guard that fails clearly if a frame isn't found, instead of timing out silently.

30. **Conflict:** Test data collides in a shared staging environment.
**Resolution:** Isolate test data per run (unique emails/IDs) or use ephemeral environments.

**Step 1:** Generate unique test data per run using a timestamp or UUID suffix.
**Step 2:** Where feasible, spin up ephemeral per-branch/per-PR environments instead of shared staging.
**Step 3:** Add a teardown step that cleans up created test data after each run.

31. **Conflict:** Session/cookie state leaks between tests.
**Resolution:** Reset the browser context per test — a fresh incognito context, not a shared browser instance.

**Step 1:** Configure the runner to create a new isolated browser context per test.
**Step 2:** Avoid reusing one logged-in browser instance across unrelated tests.
**Step 3:** Verify isolation by running tests in random order and confirming results don't change.

---

**32. Conflict:** CAPTCHA blocks automated login.
**Resolution:** Use a test-environment bypass flag or dedicated test accounts exempted from CAPTCHA.

**Step 1:** Work with security/product to add a test-env flag disabling CAPTCHA for designated test accounts/IPs.
**Step 2:** Restrict the bypass to non-production environments with audit logging.
**Step 3:** Keep one manual/exploratory check on the real CAPTCHA flow, since it's exempted from automation.

---

**33. Conflict:** Feature-flagged UI changes mid-sprint.
**Resolution:** Write tests that target both flag-off and flag-on states explicitly; don't hardcode one UI state.

**Step 1:** Identify the flag controlling the UI variant under test.
**Step 2:** Parameterize the test to run once with the flag off and once with it on.
**Step 3:** Remove the flag-off path only once the flag is fully rolled out and removed from code.

---

**34. Conflict:** Third-party embeds (ads, chat widgets) cause timeouts.
**Resolution:** Block third-party domains in the test network layer.

**Step 1:** Identify third-party domains loaded on the pages under test.
**Step 2:** Configure the test browser's network layer to block or stub those domains.
**Step 3:** Confirm the page still functions with those domains blocked, as an ad-blocker user would see it.

---

**35. Conflict:** CSS selector churn from frequent UI refactors.
**Resolution:** Use data-testid attributes decoupled from styling/structure.

**Step 1:** Add stable data-testid attributes to interactive elements in the frontend code.
**Step 2:** Migrate selectors from CSS-class/structure-based to testid-based, screen by screen.
**Step 3:** Require new interactive components to include a testid via review convention or lint.

---

**36. Conflict:** Visual regression false positives from anti-aliasing/font rendering.
**Resolution:** Use visual-diff tools with pixel-tolerance thresholds (Percy/Applitools).

**Step 1:** Adopt a visual-diff tool with a configurable pixel/perceptual tolerance threshold.
**Step 2:** Tune the threshold against a known-good baseline set to minimize false positives.
**Step 3:** Approve legitimate visual changes through the tool's baseline-update workflow, not by re-running until green.

---

**37. Conflict:** Race conditions in single-page app routing.
**Resolution:** Wait for network idle / specific DOM marker, not fixed timeout.

**Step 1:** Identify a reliable signal that a route transition is complete.
**Step 2:** Replace fixed-delay waits after navigation with a wait on that signal.
**Step 3:** Add the wait as a reusable navigation helper used by every test that triggers routing.

---

**38. Conflict:** Auth tokens expire mid-suite.
**Resolution:** Refresh/mock auth tokens via API per test instead of a UI login every time.

**Step 1:** Add an API-level login/token call at the start of each test or test group.
**Step 2:** Set the token's expiry window to cover the suite's runtime, or refresh mid-suite via a hook.
**Step 3:** Keep one true UI-login test to cover the login flow itself, since it's bypassed elsewhere.

---

**39. Conflict:** File upload/download automation behaves differently per environment.
**Resolution:** Use the framework's native file-chooser API, not OS-level dialogs.

**Step 1:** Replace OS-level file-dialog automation with the framework's native file-input API.
**Step 2:** For downloads, use the framework's download-event listener to capture the file path.
**Step 3:** Store test fixture files in the repo so upload tests don't depend on external files.

40. **Conflict:** Headless vs headed browser behavior mismatch.
**Resolution:** Run CI headless with the same engine as headed; validate parity periodically.

- **Step 1:** Run CI headless using the same browser engine version as local/headed development.
- **Step 2:** Periodically run the suite headed and diff results against headless runs.
- **Step 3:** Investigate and document headless-only failures rather than dismissing them.

41. **Conflict:** Suite execution time grows as the suite scales.
**Resolution:** Parallelize/shard tests across CI workers.

- **Step 1:** Split the suite into independent shards that don't share mutable state.
- **Step 2:** Configure CI to run shards concurrently across multiple workers.
- **Step 3:** Monitor total wall-clock time per run and re-shard as the suite grows.

42. **Conflict:** Environment config drift (staging DOM differs from prod).
**Resolution:** Parameterize base URL/config; run the same suite against each environment.

- **Step 1:** Externalize environment-specific values (URL, keys, flag defaults) into per-environment config.
- **Step 2:** Run the identical suite against each environment by swapping only config.
- **Step 3:** Alert when an environment's config diverges unexpectedly.

43. **Conflict:** WebSocket/real-time features are hard to assert.
**Resolution:** Intercept and assert on network/socket events directly, not just resulting UI state.

- **Step 1:** Use the framework's network-interception API to listen for WebSocket frames.
- **Step 2:** Assert on actual message payloads and timing, not just the resulting UI change.
- **Step 3:** Combine socket-level assertions with a final UI-state check for both angles.

44. **Conflict:** Browser extensions/ad-blockers interfere in CI.
**Resolution:** Use a clean, extension-free browser profile for automation.

- **Step 1:** Launch the automated browser with a fresh, extension-free profile every run.
- **Step 2:** Explicitly disable autofill, password-manager prompts, and similar built-in interference.
- **Step 3:** Audit the CI browser image periodically to ensure no extensions get baked in.

45. **Conflict:** Permission pop-ups (location, notifications) block the flow.
**Resolution:** Pre-set browser permissions via the CDP/framework API.

- **Step 1:** Identify which permissions the pages under test request.
- **Step 2:** Pre-grant or pre-deny them via the browser automation permission API before navigation.
- **Step 3:** Keep one test exercising the real permission-prompt UX.

46. **Conflict:** Slow selector strategies (full-path XPath).
**Resolution:** Replace with role-based or testid selectors for speed and resilience.

- **Step 1:** Audit the suite for brittle, slow full-DOM-path XPath expressions.
- **Step 2:** Replace them with role-based (ARIA) or testid-based selectors.
- **Step 3:** Add a lint rule flagging new full-path XPath usage in review.

47. **Conflict:** Analytics/tracking scripts slow page load, causing flakiness.
**Resolution:** Mock/stub analytics calls in the test environment.

- **Step 1:** Identify analytics/tracking script URLs loaded on tested pages.
- **Step 2:** Stub or block those requests in the test environment's network layer.
- **Step 3:** Keep a small separate suite validating analytics calls fire correctly, so real bugs aren't masked.

48. **Conflict:** Responsive design breaks selectors across viewport sizes.
**Resolution:** Run the suite across a defined breakpoint matrix, not just desktop.

**Step 1:** Define the app's key breakpoints (mobile, tablet, desktop) explicitly.
**Step 2:** Run the core suite at each breakpoint's representative viewport size.
**Step 3:** Fix any selector that only works at one viewport, favoring layout-independent selectors.

49. **Conflict:** Login MFA/OTP blocks automated flows.
**Resolution:** Use test accounts with MFA disabled, or a backdoor OTP-retrieval API.

**Step 1:** Provision dedicated test accounts with MFA disabled, scoped to non-production.
**Step 2:** Where MFA can't be disabled, build a test hook/API returning the current valid OTP.
**Step 3:** Keep one manual/exploratory test of the full MFA experience.

50. **Conflict:** Browser auto-updates break driver compatibility.
**Resolution:** Pin browser and driver versions in the CI image; control the upgrade cadence.

**Step 1:** Pin the exact browser and matching driver version in the CI image definition.
**Step 2:** Disable browser auto-update within that CI image.
**Step 3:** Upgrade browser and driver together on a deliberate schedule, validating the full suite first.

## C. Framework & Tool-Selection Conflicts

51. **Conflict:** Appium vs Espresso/XCUITest choice paralysis.
**Resolution:** Cross-platform E2E in Appium; platform-specific UI logic in native frameworks. Don't force one tool to do both jobs.

**Step 1:** Classify scenarios as cross-platform journey (Appium) vs platform-specific logic (Espresso/XCUITest).
**Step 2:** Assign ownership: platform teams own native suites, a cross-platform team owns Appium E2E.
**Step 3:** Document the split in the test strategy so new tests land in the right suite by default.

52. **Conflict:** Legacy Selenium suite vs migrating to Playwright.
**Resolution:** Migrate incrementally by module, running both frameworks in parallel during transition — not a big-bang rewrite.

**Step 1:** Freeze new Selenium test additions once migration begins; write new tests in Playwright.
**Step 2:** Migrate existing tests module by module, running both suites in CI during transition.
**Step 3:** Retire each Selenium module only after its Playwright equivalent runs green for a stability period.

53. **Conflict:** Low-code tool (Maestro, Katalon) vs code-first framework.
**Resolution:** Low-code for fast smoke coverage; code-first for complex, data-driven logic.

**Step 1:** Route simple smoke/UI-presence checks to the low-code tool.
**Step 2:** Route complex, data-driven, logic-heavy scenarios to the code-first framework.
**Step 3:** Review quarterly whether tests drifted into the wrong tool and move them if so.

54. **Conflict:** Different teams standardize on different frameworks.
**Resolution:** An org-level tooling council picks one primary framework per platform type, with a documented exceptions process.

**Step 1:** Form a small cross-team working group to own tooling decisions.
**Step 2:** Define one default framework per platform type with a documented exception process.
**Step 3:** Review the standard annually against the current tool landscape.

55. **Conflict:** Vendor lock-in risk with a commercial device farm.
**Resolution:** Abstract device provisioning behind a config layer so the backend is swappable.

**Step 1:** Build a thin provider layer abstracting 'get a device' from the specific vendor API.
**Step 2:** Implement the current vendor behind that interface.
**Step 3:** Periodically validate by test-running against a second vendor to confirm switching cost stays low.

**56. Conflict:** Open-source tool with weak support vs costly paid tool.
**Resolution:** Base the decision on a cost/benefit review tied to team size and maturity — not a default to “free”.

**Step 1:** List the paid tool's specific capabilities the free tool lacks.
**Step 2:** Estimate engineer time spent working around free-tool gaps vs the license fee.
**Step 3:** Decide based on team size/maturity and revisit at the next renewal cycle.

**57. Conflict:** AI self-healing locators promise less maintenance, but trust is a concern.
**Resolution:** Pilot on a non-critical suite first; measure the false-positive rate before full rollout.

**Step 1:** Enable AI locator-healing on one non-critical suite only.
**Step 2:** Track false-positive rate and maintenance-time saved over a fixed pilot period.
**Step 3:** Expand to critical suites only if pilot data shows a net improvement.

**58. Conflict:** Playwright's multi-browser claim vs real WebKit parity gaps.
**Resolution:** Validate Safari-specific behavior on real Safari/real devices; don't rely solely on Playwright's WebKit engine.

**Step 1:** Identify Safari-specific behaviors (permissions, media, WebKit-only bugs) relevant to the app.
**Step 2:** Run those specific scenarios on real Safari/real devices.
**Step 3:** Keep Playwright's WebKit for broad regression, reserving real Safari for identified edge cases.

**59. Conflict:** Cypress's weaker multi-tab/cross-origin support vs the need for it.
**Resolution:** Use Playwright for cross-origin flows; Cypress for same-origin component/E2E tests.

**Step 1:** Identify flows requiring true multi-tab or cross-origin navigation.
**Step 2:** Move those specific flows to Playwright.
**Step 3:** Keep same-origin component/E2E tests in Cypress where existing tooling/expertise is strong.

**60. Conflict:** A framework upgrade breaks the existing suite.
**Resolution:** Pin versions; upgrade in an isolated branch with full regression before merging.

**Step 1:** Pin the current framework version explicitly in the dependency manifest.
**Step 2:** Bump the version on a branch and run the full suite there before touching main.
**Step 3:** Merge only after full regression passes, documenting any breaking-change fixes.

**61. Conflict:** BDD (Cucumber/Gherkin) layer conflicts with engineers wanting plain code.
**Resolution:** Reserve BDD for business-readable acceptance criteria only; keep implementation detail out of feature files.

**Step 1:** Reserve Gherkin feature files for business-readable criteria reviewed with stakeholders.
**Step 2:** Keep step-definition implementation (selectors, waits, API calls) entirely in code.
**Step 3:** Skip BDD for purely technical/internal suites with no non-engineer audience.

**62. Conflict:** QA picks a framework without developer buy-in.
**Resolution:** Select the framework jointly with QA and engineering, since developers often maintain the suite long-term.

**Step 1:** Include engineering leads in framework evaluation from the start.
**Step 2:** Run a short trial where both QA and dev write sample tests in the candidate framework.
**Step 3:** Decide based on shared criteria: maintainability, CI integration, learning curve.

**63. Conflict:** Duplicate coverage across API, UI, and unit test layers.
**Resolution:** Apply the test pyramid; assign clear ownership per layer to avoid overlap.

**Step 1:** Map current tests to layer and flag scenarios covered redundantly.
**Step 2:** Assign ownership per layer with a documented pyramid ratio target.
**Step 3:** Remove redundant UI tests once equivalent lower-layer coverage is confirmed.

64. **Conflict:** A new tool has better DX but a smaller community.
**Resolution:** Weigh long-term maintainability against short-term productivity gain before adopting.

**Step 1:** Check release cadence, issue-response time, and dependency health.
**Step 2:** Estimate switching cost if the tool is abandoned in 1-2 years.
**Step 3:** Adopt for a bounded pilot project first if signals are mixed.

---

65. **Conflict:** Native Appium reporting is insufficient for dashboards.
**Resolution:** Integrate with a reporting layer (Allure, ReportPortal) instead of building a custom one.

**Step 1:** Choose a reporting tool that ingests Appium's output format.
**Step 2:** Wire CI to publish results to that reporting layer after each run.
**Step 3:** Retire custom in-house reporting scripts once the standard tool covers the same needs.

---

66. **Conflict:** Switching frameworks forces a full suite rewrite.
**Resolution:** Build a framework-agnostic test-data/page-object layer to ease future migrations.

**Step 1:** Build a page-object/test-data layer that doesn't directly reference framework-specific APIs.
**Step 2:** Keep a thin adapter layer translating page-object calls into the current framework's calls.
**Step 3:** When migrating, rewrite only the adapter layer, not every test.

---

67. **Conflict:** Device-farm choice (Firebase Test Lab vs BrowserStack vs Sauce Labs) is cost vs coverage.
**Resolution:** Pilot each against the actual device/OS matrix before committing to an annual contract.

**Step 1:** Define the actual device/OS matrix needed from user analytics.
**Step 2:** Run the same representative subset against each candidate vendor for a trial period.
**Step 3:** Compare cost, stability, and coverage before signing an annual contract.

---

68. **Conflict:** A consultant-introduced tool is abandoned after they leave.
**Resolution:** Require internal ownership and documentation before adopting any externally introduced tool.

**Step 1:** Require a named internal owner before adoption is approved.
**Step 2:** Require setup documentation and a maintenance runbook before the engagement ends.
**Step 3:** Review contractor-introduced tools 3-6 months after departure to confirm they're maintained.

---

69. **Conflict:** Record-and-playback tools produce brittle scripts.
**Resolution:** Use recording only as scaffolding; hand-refine selectors and waits before merging.

**Step 1:** Use record-and-playback only to scaffold a first draft.
**Step 2:** Manually replace generated selectors/waits with resilient, idiomatic equivalents.
**Step 3:** Require code review on recorded tests same as hand-written ones.

---

70. **Conflict:** Multiple assertion libraries are mixed across the suite.
**Resolution:** Standardize on one assertion library org-wide.

**Step 1:** Pick one assertion library as the org standard based on current majority usage.
**Step 2:** Migrate other libraries' usages during normal maintenance, not a dedicated rewrite.
**Step 3:** Enforce the standard via a lint rule blocking new non-standard imports.

---

71. **Conflict:** An in-house framework wrapper vs adopting the standard framework directly.
**Resolution:** Justify a wrapper only if it solves a real cross-cutting concern (reporting, retry logic); otherwise use the framework natively.

**Step 1:** List the specific cross-cutting concern the wrapper solves.
**Step 2:** Drop the wrapper if the concern isn't real or is solved by existing plugins.
**Step 3:** If justified, document the wrapper's API and keep it thin so upgrades stay easy.

---

72. **Conflict:** A test framework is incompatible with monorepo build tooling.
**Resolution:** Validate CI/build-tool compatibility before adoption, not after.

**Step 1:** Run a spike integrating the candidate framework into the monorepo's build/CI system.
**Step 2:** Confirm dependency resolution, caching, and test discovery work at monorepo scale.
**Step 3:** Adopt org-wide only after the spike proves compatibility.

73. **Conflict:** Codeless AI tools (testRigor, mabl) marketed as a full replacement vs real coverage gaps.
**Resolution:** Use AI tools for regression/smoke; keep code-first tests for complex business logic.

**Step 1:** Use codeless/AI tools for smoke and regression on stable, well-defined flows.
**Step 2:** Keep code-first tests for complex, edge-case, or data-driven scenarios.
**Step 3:** Audit AI-tool coverage against real production incidents to catch blind spots.

74. **Conflict:** Parallel-execution licensing costs scale faster than expected.
**Resolution:** Model licensing cost against suite growth before scaling parallelism.

**Step 1:** Track current parallel-execution cost per month and suite growth rate.
**Step 2:** Project cost 6-12 months out at current growth to catch surprises early.
**Step 3:** Revisit sharding strategy or licensing tier before cost becomes a forced conversation.

75. **Conflict:** Different reporting formats block a unified CI dashboard.
**Resolution:** Standardize the output format at the framework config level, org-wide.

**Step 1:** Pick one output format (e.g., JUnit XML or Allure) as the org standard.
**Step 2:** Configure every framework in use to emit that format.
**Step 3:** Point the unified dashboard at that single format, retiring custom parsers.

## D. Team, Process & Organizational Conflicts

76. **Conflict:** Mobile QA and Web QA duplicate effort on shared business logic.
**Resolution:** Own a shared API-level test suite jointly; keep UI-level suites owned per platform.

**Step 1:** Identify business logic tested identically at the UI layer by both teams.
**Step 2:** Build one shared API-level test suite owned jointly covering that logic once.
**Step 3:** Scope each team's UI suite to platform-specific rendering only, removing duplicated checks.

77. **Conflict:** Dev ships UI changes without notifying QA, breaking automation.
**Resolution:** Add a UI-change notification process, or an automated PR check that flags removed testids.

**Step 1:** Tag interactive elements with testids and add a CI check diffing them between branches.
**Step 2:** Fail or warn on the PR when a referenced testid is removed or renamed.
**Step 3:** Pair the automated check with a norm: UI-breaking changes get a QA mention in the PR.

78. **Conflict:** “Automation should replace manual QA” pressure vs real coverage gaps.
**Resolution:** Define an explicit automation vs exploratory-testing split in the test strategy; report coverage by risk area, not raw test count.

**Step 1:** Document which risk areas are covered by automation vs exploratory testing.
**Step 2:** Report coverage to leadership by risk area and defect-escape rate, not test count.
**Step 3:** Revisit the split each quarter as automation coverage genuinely expands.

79. **Conflict:** Flaky-test ownership is unclear (“not my test”).
**Resolution:** Assign clear suite ownership per team/module; track flake rate per owner.

**Step 1:** Assign each test module an owning team in the framework's config/metadata.
**Step 2:** Track flake rate per module in a CI dashboard visible to all teams.
**Step 3:** Set an SLA: a flaky test gets fixed or quarantined by its owner within one sprint.

**80. Conflict:** QA is blamed for slow releases due to a long regression suite.
**Resolution:** Invest in parallelization/tiering (smoke/regression/nightly) rather than blaming process.

- **Step 1:** Measure current suite runtime and identify the biggest time contributors.
- **Step 2:** Split into tiered suites (smoke/regression/nightly) and parallelize each tier.
- **Step 3:** Report the new pipeline timing to stakeholders with data to reset expectations.

**81. Conflict:** Automation engineers are siloed from feature teams; tests always lag.
**Resolution:** Embed automation engineers in feature squads instead of a centralized, separate QA team.

- **Step 1:** Reassign automation engineers from a central pool into feature squads.
- **Step 2:** Make test creation part of each feature's definition of done, owned within the squad.
- **Step 3:** Keep a small central guild for shared tooling/standards, separate from daily authorship.

**82. Conflict:** Conflicting priorities: new tests vs maintaining flaky ones.
**Resolution:** Allocate fixed sprint capacity (e.g., 20%) to test maintenance; don't treat it as optional.

- **Step 1:** Reserve a fixed percentage of each sprint explicitly for test maintenance in planning.
- **Step 2:** Track the flaky-test backlog as its own visible queue, not folded into general bugs.
- **Step 3:** Report maintenance time vs new coverage added each sprint to justify the allocation.

**83. Conflict:** Management wants 100% automation coverage as a KPI.
**Resolution:** Reframe the KPI around defect-escape rate or risk coverage, not raw percentage.

- **Step 1:** Start tracking defect-escape rate (bugs found in production vs caught pre-release).
- **Step 2:** Present coverage by risk area rather than a single blended percentage.
- **Step 3:** Propose the new KPI to leadership with a before/after comparison once data exists.

**84. Conflict:** Mobile release cadence (app-store review) vs web continuous deployment mismatch.
**Resolution:** Decouple release trains; use feature flags so backend/web ships independently of mobile store approval.

- **Step 1:** Wrap new features behind a server-controlled feature flag.
- **Step 2:** Ship web/backend continuously; enable the flag for mobile once the supporting app version is adopted.
- **Step 3:** Track flag cleanup as tech debt so flags don't accumulate indefinitely.

**85. Conflict:** No shared test-data strategy between mobile and web teams on the same backend.
**Resolution:** Centralize test-data-as-a-service or seeded fixtures shared across platforms.

- **Step 1:** Build or adopt a shared test-data service or seeded-fixture library both teams call.
- **Step 2:** Migrate each team's ad hoc test-data setup to the shared service incrementally.
- **Step 3:** Version the fixture definitions so both platforms rely on stable, predictable data.

**86. Conflict:** Conflicting ownership of CI pipeline infra (DevOps vs QA).
**Resolution:** Define a joint RACI: who owns pipeline config vs test content.

- **Step 1:** Draft a RACI distinguishing pipeline infrastructure from test content/config.
- **Step 2:** Review and agree the RACI with both teams' leads.
- **Step 3:** Revisit the RACI when responsibilities blur, e.g. a new tool needs both infra and test changes.

**87. Conflict:** Framework decisions are made top-down without practitioner input.
**Resolution:** Require an RFC/proposal process with engineer sign-off before mandating tools.

- **Step 1:** Require an RFC for any framework or major tooling decision, open for comment.
- **Step 2:** Set a minimum review period and require sign-off from affected team leads.
- **Step 3:** Archive RFCs so future hires can see the rationale behind current tooling.

**88. Conflict:** Offshore/outsourced automation team produces tests engineers don't trust.
**Resolution:** Pair offshore automation engineers with in-house reviewers on PRs, shared code standards.

**Step 1:** Establish a shared coding/test-writing standard referenced by both offshore and in-house engineers.
**Step 2:** Pair offshore-authored PRs with an in-house reviewer until quality is established.
**Step 3:** Rotate pairing periodically rather than permanently gatekeeping.

---

89. **Conflict:** Security team blocking test accounts/bypass flags needed for automation (MFA, CAPTCHA).
**Resolution:** Negotiate scoped, audited test-environment exceptions instead of a blanket denial.

**Step 1:** Document exactly which bypass is needed and the specific risk it introduces.
**Step 2:** Propose a scoped, audited exception limited to non-production environments/accounts.
**Step 3:** Review the exception periodically with security to confirm it's still necessary.

---

90. **Conflict:** Product team changes acceptance criteria after tests are written.
**Resolution:** Tie test cases to versioned acceptance criteria in ticket, re-review on AC change.

**Step 1:** Link each test case to a specific version/timestamp of the ticket's acceptance criteria.
**Step 2:** Require re-review of affected tests whenever the AC changes after authoring.
**Step 3:** Track AC-change frequency per team as a signal if this recurs often.

---

91. **Conflict:** Test suite runtime growing until it blocks CI merges.
**Resolution:** Enforce suite-runtime SLA, fail build (not just warn) when suite exceeds budget, triggering active pruning.

**Step 1:** Set a maximum acceptable runtime for the PR-blocking suite tier.
**Step 2:** Configure CI to fail, not just warn, when that budget is exceeded.
**Step 3:** Require teams whose additions breach the budget to optimize or move tests to a slower tier.

---

92. **Conflict:** Cross-functional conflict: performance team wants throttled network, automation team wants fast/stable runs.
**Resolution:** Separate performance-specific suite from functional automation suite, don't mix goals in one run.

**Step 1:** Split performance-focused tests into their own suite and pipeline.
**Step 2:** Keep the functional automation suite running under standard, fast, stable conditions.
**Step 3:** Schedule the performance suite separately (e.g., nightly) so it doesn't block functional CI.

---

93. **Conflict:** Legal/compliance restricts real user data in test environments vs realistic test data need.
**Resolution:** Synthetic data generation matching production schema/distribution.

**Step 1:** Profile production data's schema and statistical distribution without exposing real values.
**Step 2:** Generate synthetic data matching that profile using a data-generation library or service.
**Step 3:** Periodically validate synthetic data still resembles production closely enough to catch bugs.

---

94. **Conflict:** Automation metrics (test count, pass rate) gamed to look good vs real quality signal.
**Resolution:** Track defect escape rate and production incident correlation, not just internal pass rate.

**Step 1:** Log which production incidents had corresponding automated test coverage, and which didn't.
**Step 2:** Report defect-escape rate and this correlation alongside pass-rate/test-count metrics.
**Step 3:** Use the escape-rate trend, not raw pass rate, as the primary quality signal in retros.

---

95. **Conflict:** New hires inherit undocumented framework/tooling decisions.
**Resolution:** Maintain a living test-strategy doc with rationale for tool/framework choices.

**Step 1:** Create a test-strategy document capturing current tool choices and the reasoning behind each.
**Step 2:** Assign an owner to update it whenever a tooling decision changes.
**Step 3:** Include it in new-hire onboarding so decisions aren't relearned by word of mouth.

---

96. **Conflict:** Automation-as-safety-net vs automation-as-mandatory-release-gate expectations conflict.
**Resolution:** Explicitly define which suites block release vs which are advisory.

**Step 1:** List each test suite/tier and mark it release-blocking or advisory explicitly.
**Step 2:** Communicate the mapping to engineering and release management.

**Step 3:** Revisit the classification when a suite's reliability changes materially.

---

**97. Conflict:** Budget conflict: device farm/cloud testing cost vs perceived “we already have real devices”.
**Resolution:** Cost model comparing device farm subscription vs devices+maintenance+headcount for device lab.

**Step 1:** Calculate the fully loaded cost of an in-house device lab (hardware, replacement cycle, headcount).
**Step 2:** Compare it against device-farm subscription cost at the required matrix scale.
**Step 3:** Present both numbers side by side to decision-makers rather than debating on intuition.

---

**98. Conflict:** Escalating conflict over who fixes prod bugs missed by automation.
**Resolution:** Blameless postmortem process focused on adding regression coverage, not assigning blame.

**Step 1:** Run a postmortem for any production bug automation should plausibly have caught.
**Step 2:** Focus the output on the coverage gap that allowed it, not which individual missed it.
**Step 3:** Add the resulting regression test as the postmortem's concrete action item.

---

**99. Conflict:** Automation roadmap deprioritized every sprint for feature work.
**Resolution:** Protect automation/tech-debt time via dedicated backlog allocation agreed at planning level, not ad hoc.

**Step 1:** Negotiate a fixed percentage of each planning cycle reserved for automation/tech-debt work.
**Step 2:** Track that allocation explicitly in the backlog so skipping it is visible.
**Step 3:** Escalate to leadership with data if the protected allocation is repeatedly overridden.

---

**100. Conflict:** Diverging opinions on AI-generated test code trustworthiness.
**Resolution:** Require human review + assertion validation for AI-generated tests before merge, same as any PR.

**Step 1:** Treat AI-generated tests as a first draft requiring the same PR review as human-written code.
**Step 2:** Specifically review that assertions test meaningful behavior, not just 'runs without error'.
**Step 3:** Track defect/false-positive rates of AI-generated tests separately before relaxing review rigor.

---