CLOUDNIX TECHNICAL REPORT · CNX-TR-2026-08

Engineering Quality on Real Devices

Kiran Reddy, Lokeshwar, Avinash KS, Krishnamurthy and Avinash Nishant

Cloudnix Software Labs, Bengaluru, India

CNX-TR-2026-08 · Version 1.1 · November 2026

Abstract

Most app failures occur on devices and networks test teams do not have, and many apps are never automated because automation needs source code and script writers. We describe test platforms we have built: record and replay of Android apps from the shipped package by injecting a recording hook and generating Robotium tests from seven recorded action types; telemetry sampled inside the app under test; exploratory testing organised as a coverage grid across devices, operating systems, browsers, networks and carriers with pain and evidence per run; a severity-weighted score per build recomputed as defects are fixed; and AI generation of test cases from requirements that preserves test-data variables.

Keywords: *software testing; mobile testing; record and replay; Android instrumentation; device farm; performance telemetry; exploratory testing; quality score; AI test generation*

Contents

1. Executive summary
2. Where testing falls short
3. Record and replay without source
4. In-app telemetry
5. Exploratory coverage
6. A weighted score per build
7. AI test generation
8. Checklist: 12 questions
9. References

1 Executive summary

Most app failures happen on devices and networks that test teams do not have: budget phones, older Android versions, congested networks, memory pressure. And many apps are never automated at all, because automation needs the source code and engineers to write scripts. This report describes test platforms we have built: record and replay of Android apps from the shipped APK, without source; telemetry sampled inside the app under test; exploratory testing organised as a coverage grid across devices, operating systems, browsers, networks and carriers; a severity-weighted score per build; and test cases generated from requirements by AI.

2 Where testing falls short

Problem	Consequence	Pattern (chapter)
No source access	No automation	APK instrumentation and replay (3)
Fast lab machines	Performance issues missed	In-app telemetry (4)
Invisible exploratory coverage	Unknown gaps	Coverage grid (5)
Bug counts as readiness	Releases judged by volume, not severity	Weighted scores (6)
Slow test design	Requirements outpace tests	AI generation (7)

3 Record and replay without source



Figure 1. From a shipped APK to a replayable test.

1. Decode the APK with apktool [1]; locate the launcher activity in the manifest.
2. Copy the recording SDK in and inject a call into the launcher's `onCreate`.
3. Rebuild and sign with a debug key.
4. The SDK wraps the window callback and logs each touch: activity, view id, class, index, text and action, after a device header (model, OS version, carrier, resolution, density, app version).
5. A daemon moves completed sessions (older than 30 seconds, at least three lines) through incoming, processing and running folders.
6. A generator maps seven action types to Robotium calls [2], adds waits for activities, dialog guards and a screenshot before each assertion; the test APK is built and replayed over adb [3].

Recorded action	Generated call
Click on view	clickOnView
Click on text	clickOnText
Click in list	clickInList
Enter text	enterText
Clear text	clearEditText
Set progress	setProgressBar
Action bar item	clickOnActionBarItem

4 In-app telemetry

The injected agent samples the app's own process during tests: CPU percentage, thread count and virtual and resident memory from the process table; bytes received and transmitted by the app's UID from the traf-

fic counters [4]; and battery level from the system's battery broadcast. Samples are aligned to recorded actions, so each build has a trace per action with screenshots and device details.

Extensions. Frame timing and network shaping (2G, 3G, congested Wi-Fi) are natural additions for apps sensitive to rendering and latency.

5 Exploratory coverage

A test idea records its heuristic, oracle, model element, feature, customer coverage and classification (basic flow or deep testing), with a reviewer. A run is one idea in one environment, a combination of device (make, model, OS version, screen size), operating system, browser, network and carrier, and records pass or fail, what, why and how, perceived pain (minor, medium, high), the user's likely emotion, evidence and bug status. Plotted as ideas by environments, failures appear as a grid of marked cells. Failed runs are exported to the issue tracker in bulk with their attachments [5].

6 A weighted score per build

Checklists group functional, UX and performance checks (25 functional and UX groups, 10 performance categories) and carry a weight in the build. Failed checks are counted by severity tier:

```
checklist_score = max(0, 5 − (0.4·s4 + 0.3·s3 + 0.2·s2 + 0.1·s1))      # sN = failed checks in tier N
build_score     = Σ checklist_score × weight% / 100
```

Listing 1. Example exchange or schema.

The score is recomputed when a bug is marked fixed, so a build's readiness can be followed day by day. Packages range from essentials to release-level coverage for Android and iOS.

7 AI test generation

On the AI server, requirements selected from a knowledge base are turned into features (one parent, its child requirements) and then into test cases with steps and expected results. The pipeline maps over the knowledge base on local models and reduces per feature, preserving the project's test-data placeholders written as `{{variable}}`. Generated cases are reviewed before they enter a suite.

8 xDevice lab architecture

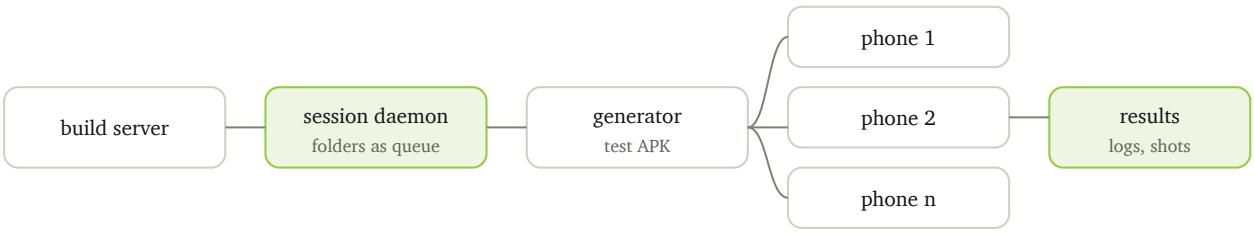

Figure 2. Lab pipeline: sessions become tests; tests run on attached phones over adb.

Phones are attached by USB to a build host and driven over adb. A folder-based queue (incoming, processing, running) makes the pipeline restartable and inspectable without a database. Each phone reports model,

OS version, carrier, resolution and density with every session, so results can be grouped by environment.

8.1 *Device selection*

Dimension	Choose by
Make and model	your users' top devices plus a budget and an old device
OS version	current, one back, oldest supported
Network	Wi-Fi, 4G, 3G or shaped equivalents
Screen	small, typical, large or tablet

9 xTest design and management

Test assets are organised as product models (elements and features), test ideas with heuristics and oracles, user flows, requirement and analysis question sets, and runs. Checklists for packages (essentials, performance, release) can be uploaded as CSV and are platform-specific for Android and iOS.

Asset	Purpose
Heuristic	a way of finding problems (e.g. interrupt, starve, overload)
Oracle	how a tester recognises a problem
Model element	the part of the product being explored
User flow	an end-to-end journey
Recommendation	the change suggested to fix a class of problem

10 xDefects, evidence and reporting

- Every failed run records what, why and how, pain level and the user's likely emotion, plus screenshots, logs and device details.
- Failures are exported to the issue tracker in bulk; attachments are uploaded per issue; tracker status is kept on the run.
- Reports are generated as HTML and PDF, and a regression email goes out per build.
- The build score and its change since the last build lead every report.

10.2 *Severity tiers*

Tier	Weight	Example
4	0.4	crash, data loss, payment failure
3	0.3	blocked flow with workaround missing
2	0.2	wrong behaviour with workaround
1	0.1	cosmetic

11 xPerformance categories

Performance is reviewed in ten categories per build: CPU, user scenario timing, memory, battery, network fluctuation, network use, file download, page weight, connections and GPU overdraw. Measured traces

come from in-app telemetry; the remaining categories are assessed by testers with captured graphs.

12 xCase study: a testing services firm

A testing services company serving app makers.

Testers needed to show clients what had been covered, not just what had failed. We built an exploratory testing platform where test ideas and runs are recorded across devices, OS, browsers, networks and carriers, with pain and evidence; a minesweeper-style grid shows coverage and failures at a glance; failures go to the client's tracker in bulk. A companion service scores startup apps per build from weighted checklists, recomputed as bugs are fixed, and a record-and-replay system turns tester sessions into automated tests without the client's source.

13 xGenerated test, annotated

```
// generated from a recorded session; one block per recorded action
solo.waitForActivity("LoginActivity");
solo.clickOnView(solo.getView("btn_signin"));
solo.enterText((EditText) solo.getView("et_phone"), "9000000000");
solo.clickOnText("Send OTP");
solo.waitForDialogToClose();
solo.takeScreenshot("login_after_otp");
solo.clickInList(2);                          // home list, third item
assertTrue(solo.waitForActivity("CartActivity"));
```

Listing 2. Example exchange or schema.

Waits are inserted before each activity change, guards around dialogs, and a screenshot before every assertion, so a failure is always accompanied by the screen that caused it.

14 xTelemetry sampling, in detail

Metric	Source	Note
CPU %	process table for the app package	per sample
Threads	process table	rising counts flag leaks
VSS, RSS	process table	memory pressure
Rx/Tx bytes	per-UID traffic counters	network cost per action
Battery	battery broadcast	drain over a session

Samples are aligned to recorded actions so that a spike is attributed to the step that caused it.

15 xScore, worked through

Checklist	Weight	Fails (t4,t3,t2,t1)	Score	Weighted
Functional	40%	1,0,1,2	$5 - (0.4+0+0.2+0.2) = 4.2$	1.68
UX	25%	0,1,2,3	$5 - (0+0.3+0.4+0.3) = 4.0$	1.00
Performance	35%	1,1,2,1	$5 - (0.4+0.3+0.4+0.1) = 3.8$	1.33
Build score				**4.01 / 5**

Fixing the tier-4 functional failure raises the functional score to 4.6 and the build score by 0.16, which is why teams fix the heaviest failures first.

16 xIntegrations

System	Integration
Issue tracker	bulk issue creation; attachments per issue; status kept on the run
Build pipeline	test APK built per session; results per build
Email	regression report per build
AI server	requirements to test cases with test-data variables

17 xImplementation roadmap

Phase	Outcome
1. Device and risk survey	Device matrix, critical flows, severity tiers
2. Lab and recording	Phones attached; first sessions recorded and replayed
3. Scoring	Checklists and weights agreed; score per build
4. Scale	Exploratory coverage, AI-generated cases, tracker integration

18 Checklist: 12 questions

1	Can apps be automated without their source?
2	Do tests run on real, varied devices?
3	Is performance measured inside the app?
4	Is exploratory coverage visible?
5	Do runs record pain and evidence?
6	Are failures exported with attachments?
7	Is readiness a weighted score, not a count?
8	Is the score recomputed as bugs are fixed?
9	Are test cases traced to requirements?
10	Are test-data variables preserved?
11	Are generated cases reviewed?
12	Who maintains the device lab?

19 xGlossary

Instrumentation	Adding code to an app to observe it.
Replay	Running recorded actions as a test.
Heuristic	A guide for finding problems.
Oracle	A way to recognise a problem.
Build score	Severity-weighted readiness of a build.

20 References

1. Apktool. *A tool for reverse engineering Android APK files*, documentation.
2. Robotium. *Robotium: Android test automation framework*, documentation.
3. Android Developers. *Android Debug Bridge (adb)*.
4. Android Developers. *TrafficStats* API reference.
5. Atlassian. *Jira Cloud REST API: bulk create issues*.
6. ISO/IEC 25010. *Systems and software Quality Requirements and Evaluation (SQuaRE): Product quality model*.

About this report. Prepared by the Cloudnix engineering team from systems we have designed, built and operated. Client and product names are withheld; illustrative values are marked as examples. Cloudnix Software Labs, Bengaluru, India · cloudnix.com. This report may be shared freely in its entirety.

Document control

Report	CNX-TR-2026-08	Version	1.1
Issued	November 2026	Status	Published
File	cloudnix-quality-engineering-v1.1.pdf		