**QA Resourcing Plan – LXP**

**Quick Reference:**

* [📊 Current Team Snapshot](#current-team-snapshot)
* [⚠️ Risks with Current Setup](#risks-with-current-setup)
* [🏋️ Proposed Dev-QA Ratio](#proposed-dev-qa-ratio-phased-plan)
* [📊 Recommended QA Mix](#recommended-qa-mix-at-12-qa)
* [📈 QA Metrics](#metrics-to-track-qa-effectiveness-post-)
* [🤖 Why AI Can’t Replace QA](#why-ai-alone-cant-replace-qa)
* [🚀 Industry References](#industry-case-references)
* [🧠 Why Hire-and-Fire Fails](#why-hire-and-fire-qa-doesnt-work)
* [📢 Ask](#ask)
* [🧪 Monthly Release Scenario](#hypothetical-scenario-qa-bottleneck-in-)
* [📅 Timeline](#timeline)

**📊 Current Team Snapshot**

| **Role** | **Count** |
| --- | --- |
| Developers | 35 |
| QA Engineers | 6 |
| Dev-QA Ratio | ~5.8:1 |
| Automation | Minimal |
| Product Stage | Early Development (high churn, unstable APIs/UI) |

**⚠️ Risks with Current Setup**

* QA bandwidth unable to match development velocity
* Regression cycles are compromised due to limited hands
* Increased probability of bugs escaping to production
* QA team burnout and lack of breathing space
* Delayed feedback loops causing bottlenecks in CI/CD pipeline

**🏋️ Proposed Dev-QA Ratio (Phased Plan)**

| **Phase** | **Target Ratio** | **QA Headcount** | **Rationale** |
| --- | --- | --- | --- |
| Immediate Fix | 4.5:1 | 8 QA | Relieves current pressure, improves test turnaround |
| Within 1-2 months | 3:1 | 12 QA | Enables parallel QA per stream, ensures test coverage across modules |
| Stabilized Phase | 4:1 or 5:1 | 9-10 QA | Regression streamlined, QA focuses on UAT and release checks |

**📊 Recommended QA Mix (at 12 QA)**

* **10 Functional QA**: Manual testing, regression, UAT, exploratory
* **1 QA Lead**: Strategy, metrics, release ownership
* **1 QA Support/Coordination**: Triage, documentation, tools management

**📈 Metrics to Track QA Effectiveness Post-Ramp-up**

| **Metric** | **Target/Goal** |
| --- | --- |
| Test Coverage | >85% on core workflows |
| Mean Time to Detect (MTTD) | <1 day |
| Escaped Defects (per release) | <3 major bugs |
| QA Regression Cycle Duration | <2 working days |

**🤖 Why AI Alone Can't Replace QA**

**🛠️ Current State of AI Tools in QA**

| **Tool Type** | **Common Examples** | **Limitations** |
| --- | --- | --- |
| AI Test Case Generation | Testim, mabl, Functionize | Limited context awareness, overfit to UI structure |
| Visual Regression | Applitools, Percy | Struggles with UX intent or business logic shifts |
| Scriptless Automation | TestSigma, Katalon Studio | Fragile in evolving product environments |
| Code Assistants | GitHub Copilot, Tabnine | Still need QA oversight and validation |

Despite rapid progress, AI tools still fall short of fully replacing human QA, especially in new or fast-changing products. AI-driven test generation and bug detection are promising, but:

* AI cannot reason through business logic, subjective UI/UX quality, or evolving domain rules.
* In volatile environments, AI-generated tests often become brittle or misaligned.
* Human QA brings contextual understanding, creative edge case exploration, and cross-functional collaboration.
* Most AI tools still require QA engineers to curate, validate, and maintain them.

In fact, increased complexity and velocity make the **need for strategic QA stronger**, not weaker.

**🚀 Industry Case References**

**Case 1: Atlassian QA Model**

* Atlassian promotes a 'quality assistance' model where developers own quality, but still maintain dedicated QA roles during product growth stages.
* Initially had 3:1 ratio; grew QA teams temporarily to 2:1 when launching major platform shifts.
* Source: Atlassian Engineering Blog, "Quality at Speed" Series

**Case 2: ThoughtWorks Client Implementation (Retail Sector)**

* Project started with a dev-heavy approach, leading to missed regressions.
* Rebalanced QA staffing to 1 QA per 3-4 developers.
* Enabled 70% reduction in post-release defect rate and 30% faster regression cycles.
* Source: ThoughtWorks Technology Radar – QA Practices (2021 Edition)

**Case 3: Google Testing Strategy Evolution**

* Emphasized developers taking responsibility for testing, but not in isolation.
* While early blogs like "Just Say No to More Testers?" suggested aggressive developer-led testing, later implementations acknowledged the challenges of complex products and reintroduced strategic QA roles.
* In practice, zero-QA setups did not scale well for aggressive release models, and Google evolved toward hybrid roles.
* Source: Google Testing Blog (Whittaker), industry retrospectives

**🧠 Why 'Hire-and-Fire' QA Doesn’t Work**

A short-term approach to hiring QA for immediate needs and offloading them post-release may seem cost-effective but has long-term drawbacks:

* **QA effectiveness improves with domain familiarity**; new hires take 4–6 weeks just to become productive.
* Constant churn leads to **loss of tribal knowledge**, missed edge cases, and inconsistent test coverage.
* Building **test scenarios, understanding platform complexity, and syncing with devs** require continuity.
* Stable QA teams enable **faster onboarding of new features**, long-term ownership of test data, and reusable regression packs.

**📊 Supporting Data Points:**

* Studies show that **onboarding a new QA can take 4–8 weeks** before full contribution.
* Retained QA engineers **reduce average escaped bugs per release by 40–60%** over time.
* High-churn QA teams **spend 25–30% of their time revalidating test assumptions** already known to longer-tenured teams.
* Business-critical flows tested by experienced QA are **3x more resilient** in catching corner cases.

True product quality stems from **consistency and context**, not contractual headcount. A short-term approach to hiring QA for immediate needs and offloading them post-release may seem cost-effective but has long-term drawbacks:

* **QA effectiveness improves with domain familiarity**; new hires take 4–6 weeks just to become productive.
* Constant churn leads to **loss of tribal knowledge**, missed edge cases, and inconsistent test coverage.
* Building **test scenarios, understanding platform complexity, and syncing with devs** require continuity.
* Stable QA teams enable **faster onboarding of new features**, long-term ownership of test data, and reusable regression packs.

True product quality stems from **consistency and context**, not contractual headcount.

**📢 Ask**

* Approve QA ramp-up from 6 to 12 over 2 months
* Immediate onboarding of 2-3 QA to avoid testing bottlenecks
* Align hiring with product velocity and release goals

**🧪 Hypothetical Scenario: QA Bottleneck in Monthly Releases**

**Setup:** 35 Developers, 6 QA, Monthly Release (1st–30th)

**Flow:**

* Day 1–15: Devs build 30+ stories
* Day 16–20: Code freeze, features handed to QA
* Day 21–27: All features expected to be tested
* Day 28–30: Final regression + sanity

**Breakdown:**

* 6 QA test ~30 stories in <7 days
* Delayed feature handoffs prevent parallel QA
* Critical stories consume QA bandwidth, delaying others
* Regression cycles are rushed or skipped

**Result:**

* QA burnout (9–10 hr days), unstable builds, missed issues
* Dev idle time while QA catches up, or QA skips ahead
* Stakeholder trust erodes: "It passed QA but failed in prod"

Even with over-effort, QA remains reactive. A 3:1 ratio (12 QA) ensures smoother handoffs, test coverage, and release stability.

**📅 Timeline**

| **Milestone** | **Date** |
| --- | --- |
| Initial QA Hire (2-3 QAs) | This Month |
| Complete Team (12 QAs) | +60 Days |
| Regression < 2 Days | +75 Days |

*Prepared by: Kishore Murkhanad* *Role: QA Manager* *April 2025*