# Evaluating Performance Across the Software Development Lifecycle

Modern SDLC demands performance to be a **continuous concern**, not an afterthought. Incorporating **Shift Left (early)** and **Shift Right (post-release)** performance strategies ensures resilience, speed, and scalability throughout.

**1️⃣ Requirements & Design Phase *(Shift Left)***

🎯 *Prevent* performance issues through thoughtful architecture and well-defined goals.

**🔹 Key Activities & Examples:**

* **Define Non-Functional Requirements (NFRs):**  
  *Example:* “Checkout API must respond in <2 seconds for 95% of 2,000 concurrent users.”
* **Architecture & Design Review:**  
  *Example:* Identify a potential bottleneck where a cart service calls inventory service synchronously → recommend asynchronous messaging with retry logic.
* **Technology Stack Evaluation:**  
  *Example:* Flag potential performance risks in using a client-side-heavy JavaScript SPA on low-end mobile devices.
* **Initial Workload Modeling:**  
  *Example:* Black Friday peak = 10,000 users; 60% browsing, 30% adding to cart, 10% checking out.

💡 *Why It Matters:* Prevents foundational flaws, like poor database design or underpowered infrastructure, from surfacing later when they’re harder to fix.

**2️⃣ Development & Unit Testing Phase *(Shift Left)***

🎯 Ensure individual code components perform efficiently.

**🔹 Key Activities & Examples:**

* **Performance Unit Tests:**  
  *Example:* Time a loop that calculates discounts for 1,000 items and optimize it to use caching.
* **Static Code Analysis:**  
  *Example:* Tool flags an N+1 query issue in product listing – fixed by restructuring SQL join.
* **Code Review for Anti-Patterns:**  
  *Example:* A senior dev catches repeated API calls in a loop instead of batching requests.
* **Micro Benchmarking:**  
  *Example:* Track time taken for image resizing logic before/after using a faster compression library.

💡 *Why It Matters:* Minor inefficiencies at the function level can snowball into major system bottlenecks when scaled.

**3️⃣ Integration & System Testing Phase *(Mid-Development / Shift Left)***

🎯 Validate early-stage performance as components start to interact.

**🔹 Key Activities & Examples:**

* **CI-Based Performance Tests:**  
  *Example:* Jenkins runs a k6 test with 100 users on every major PR merge → detects new 500ms delay in checkout API.
* **Component-Level Load Testing:**  
  *Example:* Stress the search microservice with 1,000 queries/minute to observe memory use.
* **Environment Setup (Staging):**  
  *Example:* Replica of production with masked user data to allow realistic tests.
* **Script Development:**  
  *Example:* Automate login → browse → add-to-cart → checkout flow with JMeter and parameterized user data.

💡 *Why It Matters:* Catches cross-service latencies, shared resource contention, and integration misconfigurations.

**4️⃣ UAT / Pre-Production Phase *(Traditional Performance Testing)***

🎯 Validate the entire application under realistic, peak, and edge-case loads.

**🔹 Key Activities & Examples:**

* **Load Testing:**  
  *Example:* Simulate 5,000 concurrent users on the full end-to-end e-commerce flow.
* **Stress Testing:**  
  *Example:* Push beyond expected load (e.g., 15,000 users) to identify failure thresholds.
* **Endurance Testing:**  
  *Example:* Run login → browse → checkout flow continuously for 8 hours to detect memory leaks.
* **Scalability Testing:**  
  *Example:* Test with and without autoscaling enabled to assess horizontal scalability in AWS.
* **Monitoring & Analysis:**  
  *Example:* Use Grafana dashboards to correlate TPS drops with CPU spikes and pinpoint a memory bottleneck.

💡 *Why It Matters:* Confirms whether the application meets SLA and business goals before going live.

**5️⃣ Production / Post-Deployment Phase *(Shift Right)***

🎯 Monitor, learn, and continuously optimize in real-world conditions.

**🔹 Key Activities & Examples:**

* **Real User Monitoring (RUM):**  
  *Example:* Track page load times across different geographies – Southeast Asia sees 2x slower speeds → investigate CDN setup.
* **Synthetic Monitoring:**  
  *Example:* Scheduled k6 scripts from 5 cities worldwide simulate checkout every 5 minutes to monitor uptime and latency.
* **APM Integration (e.g., Datadog, Dynatrace):**  
  *Example:* Detects sudden CPU spike due to a bad deployment → alert triggered, rollback initiated.
* **A/B Testing or Canary Releases:**  
  *Example:* New recommendation algorithm rolled out to 5% users shows 300ms slower response → rollback before global impact.
* **Chaos Engineering:**  
  *Example:* Use Gremlin to shut down a node randomly to test service failover and load redistribution.
* **Feedback Loop:**  
  *Example:* Postmortem reveals cart latency under load → updates backlog with “optimize DB index on cart\_items table.”

💡 *Why It Matters:* Production is the ultimate test bed. Continuous feedback ensures better future planning and customer trust.

**📊 SDLC-Based Performance Evaluation Summary**

| **SDLC Phase** | **Key Focus** | **Example Activity** |
| --- | --- | --- |
| Requirements & Design | Prevention, Architectural Readiness | Define SLA: Login <2s; Review DB schema for normalization issues |
| Development | Efficient Code, Micro Benchmarks | Catch unoptimized loops, measure single-function response times |
| Integration/System | Inter-component Stress, CI Integration | Run service-level load test on microservices |
| UAT / Pre-Prod | Full-Scale, Peak & Failover Testing | Endurance test for 8 hrs, load test with 10K users |
| Production | Real-World Monitoring & Feedback | APM dashboards, Synthetic monitors, Canary rollout observation |

**🚦 Final Thoughts**

Integrating performance evaluation across the SDLC:

✅ Reduces risk  
✅ Improves user experience  
✅ Cuts rework costs  
✅ Enables continuous improvement

🔁 *Performance is not a phase—it's a culture.*