SDLC vs STLC — Practical Differences (Document Version)

Note: You wrote “SDLC and SDLC.” This document assumes you meant SDLC (Software Development Life Cycle) and STLC (Software Testing Life Cycle). No tables, no interview tone—just clear, structured explanation you can paste into a report.

# 1) What They Are

* SDLC is the end‑to‑end process for building and shipping software: from an idea to a running system in production and its ongoing maintenance.
* STLC is the end‑to‑end process for assuring quality and managing risk within that software: planning, designing, executing, and improving tests across the lifecycle.

In short: SDLC builds value; STLC proves that value is reliable, secure, and usable.

# 2) Why Both Matter

* SDLC without STLC risks shipping fast but breaking trust—bugs, security holes, poor performance.
* STLC without SDLC context becomes checkbox testing—cases may pass but miss business value.

They work together: SDLC sets what to build and when; STLC shapes how we measure quality and decide if it’s safe to ship.

# 3) Where They Sit in a Project Timeline (Narrative)

As requirements form, SDLC captures scope and design choices. At the same time, STLC runs a risk workshop and drafts a test strategy.

As code is written, SDLC focuses on implementation and integration; STLC prepares data, environments, mocks/stubs, and builds automation into CI.

As release nears, SDLC plans deployment and rollback; STLC executes functional, performance, security, and accessibility checks and issues a quality report.

After release, SDLC monitors product KPIs; STLC analyzes incidents/escaped defects and improves coverage and flakiness for the next cycle.

# 4) Roles and Responsibilities

* SDLC: product managers, business analysts, architects, developers, DevOps/SRE, security engineers, QA.
* STLC: QA/test engineers lead quality planning and execution, partnering with product, dev, DevOps, and security.

Ownership differs, but collaboration is tight—quality gates are shared responsibilities.

# 5) Entry and Exit Criteria (Gates)

* SDLC entry: a validated business need and prioritized backlog. Exit: feature released with monitoring and runbooks in place.
* STLC entry: stable-enough requirements or acceptance criteria to design tests. Exit: agreed‑upon coverage, severity thresholds, pass rates, non‑functional targets (e.g., latency, error rate) are met, and a go/no‑go sign‑off is recorded.

# 6) Artifacts and Deliverables

* SDLC produces: PRD/user stories, architecture/design docs, source code, CI/CD pipelines, infra definitions, release notes, runbooks.
* STLC produces: test strategy/plan, traceability (requirements ↔ tests), test cases and data, automation suites (unit/integration/API/UI), defect reports, dashboards, and a final quality summary.

# 7) Metrics and KPIs (Different Lenses)

* SDLC focuses on flow and reliability of delivery: lead time, deployment frequency, change failure rate, mean time to restore (MTTR).
* STLC focuses on quality signals: requirements coverage, automated coverage, pass/fail trends, defect density and escape rate, flake rate, and non‑functional baselines like p95 latency and error rate under load.

# 8) Tools and Environments (Mindset)

* SDLC wiring: Git, code review, CI/CD (e.g., Jenkins, GitHub Actions, GitLab), containers/Kubernetes, infrastructure‑as‑code, observability.
* STLC validation: test management (Xray/Zephyr/TestRail), frameworks (Selenium/Playwright/Cypress for UI; REST Assured/Postman for API), performance (JMeter/LoadRunner), security (OWASP tooling), accessibility, and mocking/stubbing tools.

The point isn’t the brand names, but that SDLC wires the factory; STLC validates the output of that factory continuously.

# 9) Common Pitfalls and How to Avoid Them

* SDLC pitfall: vague acceptance criteria lead to rework. Avoid by writing testable acceptance criteria and aligning on success measures.
* STLC pitfall: only happy‑path testing. Avoid by systematic risk analysis, boundary/negative tests, and data variety.
* Joint pitfall: late testing. Avoid by shifting left—embed test design and automation during design/build, not after.

# 10) Short Walkthrough Example (Narrative, No Tables)

Feature: “Schedule a bank transfer.”

— SDLC defines the user story, cut‑off times, time‑zone rules, and integration to payment rails; architects choose APIs and data model; dev builds services/UI; ops prepares deployment and rollback.

— STLC identifies risks (cut‑off windows, $0 amounts, cross‑time‑zone scheduling), writes scenarios and boundary tests, seeds data, sets up mocks for the bank rails, builds API‑first automation and then UI checks, runs performance and security checks, and publishes a quality report for go/no‑go.

If production shows an issue (e.g., time‑zone conversion bug), SDLC triages and patches; STLC adds a targeted regression test to prevent recurrence.

# 11) Quick Recap — Key Differences (Concise)

* Purpose: SDLC builds product value; STLC measures and protects that value.
* Scope: SDLC spans ideation to operations; STLC spans quality planning to closure across the same timeline.
* Ownership: SDLC is multi‑discipline; STLC is QA‑led with cross‑team input.
* Gates: SDLC ships value; STLC signs off risk and readiness.
* Evidence: SDLC delivers software and runbooks; STLC delivers test assets, defect intel, and a quality summary.