SDLC Phases & Responsibilities Checklist

A Practical Guide for Software Project Teams

1. Introduction

The Software Development Life Cycle (SDLC) is a structured process that guides teams through the planning, creation, testing, deployment, and ongoing support of software applications. Following a well-defined SDLC helps organizations deliver high-quality software efficiently, manage risks, and ensure alignment with business goals.

This checklist is designed to empower teams by outlining key best practices, mapping roles and responsibilities, and highlighting actions to mitigate common risks. Whether you're planning a new project or managing an ongoing initiative, use this guide to ensure every phase is well-executed and nothing critical is overlooked.

You can use this checklist as a step-by-step guide throughout each SDLC phase, or as a quick reference during project planning meetings. Teams may also adapt the checklist to fit their unique workflows, ensuring consistent standards are maintained across projects.

2. SDLC Phases & Responsibilities Checklist

Below, each SDLC phase is broken down with responsible roles and actionable items. Use this as a reference table or adapt the bullet points for your team's specific needs.

2.1 Planning & Requirements Gathering

- **Responsible Roles:** Product Owners, Business Analysts, Stakeholders
- **Checklist:**
  - Document scope, goals, and acceptance criteria ✓

- *Example:* Capture project objectives in a requirements document and specify what defines success for each feature.
- Prioritize features based on business value ✓
- *Example:* Use MoSCoW prioritization (Must-have, Should-have, Could-have, Won't-have) to focus development on essential deliverables.
- Validate requirements with stakeholders ✓
- *Example:* Hold stakeholder workshops or review sessions to confirm understanding and buy-in.
- Identify dependencies and risks ✓
- *Example:* Map out integrations with external systems and flag potential delays due to third-party dependencies.

2.2 System Design

- **Responsible Roles:** Architects, Senior Engineers
- **Checklist:**
    - Document architecture and APIs ✓
    - *Example:* Create system diagrams and write API specifications to guide development.
    - Define non-functional requirements (security, scalability, compliance) ✓
    - *Example:* Specify encryption standards, expected user load, and regulatory requirements like GDPR.
    - Review design iteratively ✓

- ◦ *Example:* Schedule design review meetings at major milestones to refine solutions.
- ◦ Approve architecture decisions ✓
- ◦ *Example:* Document final architecture choices and obtain sign-off from key stakeholders.

2.3 Development & Iteration

- **Responsible Roles:** Developers, DevOps (for CI/CD integration)
- **Checklist:**
    - Follow coding standards and peer review ✓
    - *Example:* Use code linters and require pull request reviews before merging.
    - Implement automated unit tests ✓
    - *Example:* Write unit tests for every new module and integrate them in the build pipeline.
    - Ensure integration with CI/CD pipelines ✓
    - *Example:* Set up Jenkins or GitHub Actions to build and test code automatically on every commit.
    - Apply feature toggles for safe releases ✓
    - *Example:* Use flags to enable or disable new features in production without redeploying.

2.4 Testing & Software Validation

- **Responsible Roles:** QA Engineers, SREs

- **Checklist:**
    - Conduct unit, integration, and end-to-end testing ✓
    - *Example:* Test components individually, then together, and finally in real-world scenarios.
    - Validate acceptance criteria ✓
    - *Example:* Compare test results against the original requirements to ensure all conditions are met.
    - Shift-left testing implemented ✓
    - *Example:* Start testing activities early in development to catch defects sooner.
    - Log defects and track resolution ✓
    - *Example:* Use Jira or Azure DevOps to report bugs and monitor fixes throughout the cycle.

2.5 Deployment & Release

- **Responsible Roles:** DevOps, Developers
- **Checklist:**
    - Verify deployment automation pipelines ✓
    - *Example:* Test automated scripts for repeatable and error-free deployments.
    - Plan rollback strategies ✓
    - *Example:* Prepare procedures to revert releases quickly if issues arise.
    - Conduct canary/blue-green deployments ✓

    - *Example:* Release updates to a small user group or alternate environment before full launch.
    - Monitor post-release metrics ✓
    - *Example:* Track usage, errors, and performance to ensure the release is stable.

2.6 Maintenance

- **Responsible Roles:** Support, SREs, Product Teams
- **Checklist:**
    - Monitor system health and logs ✓
    - *Example:* Use monitoring tools like Datadog, Splunk, or Azure Monitor to detect issues early.
    - Schedule security patches and updates ✓
    - *Example:* Set up a regular cadence for updating dependencies and fixing vulnerabilities.
    - Track technical debt ✓
    - *Example:* Maintain a backlog of items that need refactoring or cleanup.
    - Conduct post-mortems for incidents ✓
    - *Example:* After outages or bugs, perform root cause analysis and document lessons learned.

2.7 Summary Table: SDLC Phases & Responsibilities

Phase	Roles	Key Actions

Planning & Requirements	Product Owners, Business Analysts, Stakeholders	Document scope, goals, and acceptance criteria Prioritize features Validate requirements Identify dependencies and risks
System Design	Architects, Senior Engineers	Document architecture and APIs Define non-functional requirements Review design iteratively Approve architecture decisions
Development & Iteration	Developers, DevOps	Follow coding standards Implement automated unit tests CI/CD integration Apply feature toggles

Testing & Validation	QA Engineers, SREs	Conduct all levels of testing Validate acceptance criteria Shift-left testing Log defects and track resolution
Deployment & Release	DevOps, Developers	Verify automation pipelines Plan rollbacks Canary/blue-green deployments Monitor metrics
Maintenance	Support, SREs, Product Teams	Monitor health and logs Schedule security patches Track technical debt Post-mortems for incidents

By following this checklist, teams can ensure every stage of the SDLC is managed effectively, responsibilities are clear, and risks are proactively addressed—leading to successful software projects.

3. Best Practices Quick Reference

- **Automate builds, tests, and deployments:** Use CI/CD pipelines to reduce manual intervention, minimize errors, and speed up delivery cycles.
- **Shift security and quality left:** Integrate security checks and quality validation early in the development process to catch issues sooner and lower remediation costs.
- **Small, iterative releases:** Deliver changes in manageable increments to enable faster feedback, reduce risk, and simplify rollbacks if needed.
- **Define exit criteria for each phase:** Clearly establish what must be completed before moving to the next SDLC phase, ensuring consistent quality and readiness.
- **Use telemetry and observability:** Implement monitoring, logging, and tracing to gain real-time insights into system health, performance, and user behavior.
- **Protect the deployment pipeline:** Secure your CI/CD tools and processes with access controls, audit trails, and regular reviews to prevent unauthorized changes or breaches.

4. Metrics & Success Tracking

- **Flow Metrics:**
  - **Lead time:** The time from code commit to production deployment, reflecting overall delivery speed.
  - **Deployment frequency:** How often new releases are deployed, indicating agility and responsiveness.
  - **Change failure rate:** The percentage of deployments causing incidents or requiring hotfixes, highlighting release quality.
  - **MTTR (Mean Time to Recovery):** The average time to restore service after a failure, measuring operational resilience.
- **Quality Metrics:**
  - **Escaped defects:** Number of bugs found in production after release, pointing to gaps in earlier testing phases.
  - **Test coverage:** Percentage of code exercised by automated tests, serving as a proxy for potential defect detection.
  - **Validation results:** Outcomes of acceptance and regression testing, demonstrating readiness and stability.
- **Business Metrics:**
  - **Cycle time:** The duration from idea to customer delivery, reflecting process efficiency.
  - **Customer satisfaction:** Feedback and satisfaction scores from users, measuring business value and success.

5. Career & Skill Development

- **Recommended Certifications:** Pursuing credentials like Certified SDLC Professional or the GSDC SDLC certification can validate your expertise in software lifecycle management and demonstrate commitment to best practices.
- **Role-Specific Development Tips:**
    - **DevOps:** Focus on continuous integration, automation, and cloud platform proficiency to stay current in fast-evolving environments.
    - **QA:** Expand your skills in automated testing frameworks, performance testing, and security validation to drive higher product quality.
    - **Project Management:** Grow expertise in Agile methodologies, risk management, and stakeholder communication for effective leadership.
    - **Architecture:** Deepen your knowledge in scalable design, API management, and emerging technologies such as microservices and containerization.

6. Tips for Using the Checklist

- Regularly update the checklist to reflect the needs of each project and phase, ensuring it remains relevant as the team evolves.
- Assign clear ownership to each checklist item, making accountability and progress transparent to all stakeholders.

- Review and adjust the checklist based on lessons learned during retrospectives; use feedback to continuously improve the process and address gaps.

7. Conclusion

The SDLC should be embraced as a living process that adapts to your project's unique needs and organizational growth. Iterative improvement and regular reflection help teams stay resilient and deliver better outcomes over time. For deeper learning and practical examples, visit our blog or explore the recommended resources to expand your understanding and apply these principles to your own workflows.

GSDC
Global Skill Development Council

CERTIFIED SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC) PROFESSIONAL

CERTIFIED SDLC PROFESSIONAL VALIDATES YOUR EXPERTISE IN MANAGING AND OPTIMIZING SOFTWARE DEVELOPMENT FOR QUALITY AND EFFICIENCY.

GSDC
Global Skill Development Council
Software Development Life Cycle Professional
CERTIFIED

ABOUT GSDC CERTIFICATION

LIFETIME VALIDITY

GSDC Certification is an globally accreditted certification with lifetime validity.

EBOOK

Extensive and exclusive Ebook created by world's experts to help you with understanding core concepts.

CREATED BY EXPERTS

GSDC certifications are created and authored by world's leading experts in the field.

LEARNING MATERIALS

Get access to learning materials such as videos, ebooks, templates, and practice exams, which will help you clear the certification exam.

LEARNING OBJECTIVE

- Enhances development visibility for all stakeholders
- Improves planning, scheduling, and estimating efficiency
- Strengthens risk management and cost estimation decisions

Enroll now with the code **LEARN20** To avail **20%** discount

Enroll Now

www.gsdcouncil.org