# **Introduction**

The Systems Development Life Cycle (SDLC) serves as a fundamental methodology for designing and implementing information systems in a structured and systematic manner. This framework guides the entire process of system development—from initial planning and analysis to design, implementation, and ongoing maintenance—ensuring that solutions are developed efficiently, meet organizational needs, and remain adaptable in an ever-changing technological landscape.

In today's fast-paced digital environment, businesses and users demand systems that evolve alongside their requirements. The SDLC provides a disciplined approach to managing this evolution, ensuring that each phase—**Planning, Analysis, Design, Implementation, and Maintenance**—is executed with clear objectives, defined processes, and measurable outcomes. Each stage involves specialized functions, tools, key personnel, and deliverables that collectively contribute to the successful creation and long-term sustainability of robust information systems

# GENERIC PHASES OF SYSTEM DEVELOPMENT LIFE CYCLE

# 

## **PLANNING**



## Main Functions

The main functions include defining the project scope and objectives by outlining what the software or system aims to achieve—such as solving a specific problem or improving efficiency—while establishing a detailed description of its features, functionalities, and boundaries to prevent unnecessary additions later. Additionally, feasibility studies are conducted to assess the project's viability, including technical feasibility (whether the desired functionality can be implemented with available technology and resources), economic feasibility (whether the project is cost-effective and offers a good return on investment), and operational feasibility (whether the software will integrate well with existing systems and workflows). These steps ensure the project is well-defined, practical, and aligned with business goals..

## KEY INPUTS

Budget constraints play a critical role in project planning, as they involve estimating the costs associated with software development, deployment, and ongoing maintenance. This includes expenses such as labor (developer salaries, project management), infrastructure (servers, cloud services), licensing fees (third-party tools, APIs), and future updates or bug fixes. Proper budget planning ensures financial feasibility and prevents cost overruns that could derail the project.

Additionally, selecting the right tools and technologies is essential for efficient development and long-term sustainability. This includes choosing appropriate **programming languages** (such as Python for AI, JavaScript for web apps, or Java for enterprise systems), **frameworks** (like React for front-end, Django for back-end, or TensorFlow for machine learning), and **infrastructure** (cloud platforms such as AWS or Azure, containerization tools like Docker, or CI/CD pipelines for automation). The right technology stack ensures scalability, performance, and ease of maintenance while aligning with the project’s technical requirements and team expertise

## **MAJOR TOOLS**

Project management utilizes various tools to organize and track work effectively. **Gantt charts** provide a visual timeline of tasks, durations, and progress. **Kanban boards** help teams visualize workflows using columns and task cards, ideal for Agile projects. Simple yet powerful, **task lists** track assignments and deadlines at an individual level. For strategic alignment, **project roadmaps** outline high-level goals, timelines, and key milestones. Finally, **tree diagrams (Work Breakdown Structures)** break complex projects into smaller, manageable components, improving clarity and execution. Together, these tools enhance planning, coordination, and efficiency in project delivery.

## Major Techniques

Effective planning is essential for organizational success, and various techniques help streamline this process. \*\***Operational planning\*\*** ensures that all business functions align with overarching organizational goals, optimizing efficiency and coordination. \*\***Contingency planning\*\*** prepares businesses for unforeseen disruptions by identifying risks and developing proactive response strategies. For project management, \*\***critical path analysis\*\*** pinpoints the most essential tasks, ensuring timely completion by focusing on key activities. Additionally, \*\***SWOT analysis\*\*** evaluates internal strengths and weaknesses alongside external opportunities and threats, enabling data-driven strategies that leverage advantages while addressing vulnerabilities. Together, these techniques enhance decision-making, risk management, and long-term planning.

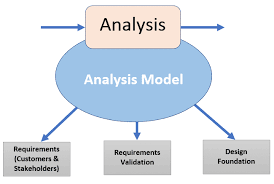
## Specialists

Successful project planning relies on collaboration between key team members, each contributing unique expertise. The **Project Manager** leads the planning efforts by defining the project scope, objectives, and timelines while ensuring alignment with business goals. The **Business Analyst** acts as a bridge between stakeholders and the technical team, gathering and analyzing requirements to ensure clarity and feasibility. **Developers** provide critical technical insights during planning, assessing feasibility, potential challenges, and resource needs. Finally, **Stakeholders,** including clients and end-users, offer essential input on project goals, expectations, and requirements to ensure the final deliverable meets business needs. Together, these roles create a structured and well-informed planning process.

## Key outputs

Two critical components of project planning include the Project Charter and the Timeline & Milestones. The **Project Charter** serves as a foundational high-level document that formally authorizes the project, defining its purpose, objectives, key stakeholders, and overall scope. It aligns the team and stakeholders on the project’s vision and expectations. Meanwhile, the **Timeline & Milestones** provide a structured schedule, breaking down the development process into phases with specific deadlines and key deliverables. This ensures progress tracking, keeps the team accountable, and helps identify potential delays early. Together, these elements establish clarity, direction, and measurable progress for successful project execution.

### **ANALYSING**



### Main Functions

The analysis phase serves two key functions. First, it produces detailed **requirements documentation** that clearly specifies the system's functionality, behavior, and performance expectations by gathering and analyzing stakeholder input. This documentation becomes the project's blueprint. Second, it conducts a **feasibility analysis** evaluating four critical aspects: technical (can we build it with available technology?), operational (will it work with current processes?), economic (is it cost-effective?), and scheduling (can we deliver on time?). Together, these ensure the project is well-defined and viable before development begins.

### KEY INPUTS

The analysis phase relies heavily on important sources of information to properly define the system’s needs. One major input is the **documentation of existing systems**, which provides detailed records of how the current processes and technologies operate. Reviewing these documents helps identify gaps, inefficiencies, and areas for improvement. Another key input is the **observation of current workflows**. By watching how employees and systems work in real-world conditions, analysts can better understand daily challenges and identify steps that could be streamlined or automated. **Stakeholder and end-user interviews** are also critical during this phase. These interviews allow analysts to hear directly from those who use the system or are affected by it, ensuring that the final product addresses real needs rather than assumptions. Collectively, these inputs help create a complete picture of what the organization requires from the new system.

### **MAJOR TOOLS**

The analysis phase produces several important deliverables that help define system functionality and user interactions. **Dataflow diagrams** visually map how information moves between different system processes, clarifying relationships and workflows. **Software Requirements Specifications (SRS)** provide a comprehensive document detailing both functional needs (what the system should do) and non-functional requirements (performance, security, etc.). **Use cases** illustrate real-world scenarios, demonstrating how various users will engage with the system step-by-step. Additionally, **user stories** offer concise, end-user perspectives in simple narrative form (e.g., "As a customer, I want to reset my password so I can regain account access"). Together, these artifacts ensure clear communication of system behavior, user needs, and technical expectations before development begins ( often used in agile development)

### MAJOR TECHNIQUES

Several key methods help teams effectively collect and analyze requirements. **Interviews** allow for in-depth, one-on-one discussions with stakeholders to uncover detailed insights. **Prototyping** involves creating simple wireframes or mockups to visualize how requirements translate into actual user interfaces and experiences. **Questionnaires** enable teams to efficiently gather feedback from a broad audience through structured surveys. **Observations** involve studying end-users as they interact with existing systems to identify pain points and workflow challenges. Finally, **workshops** bring together stakeholders and development teams for collaborative sessions that align everyone on requirements and expectations. Together, these techniques ensure a well-rounded understanding of user needs and system goals.

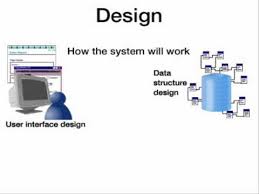
### SPECIALISTS

Several key specialists and roles contribute to the success of the analysis phase in software development. **Business Analysts** act as the bridge between stakeholders and the technical team, gathering business needs, assessing feasibility, and documenting clear, actionable requirements. **Project Managers** oversee the entire process, ensuring alignment with business goals, timelines, and budgets while keeping the project on track. **System Analysts** collaborate with Business Analysts to evaluate technical feasibility, ensuring proposed solutions adhere to project constraints. Additionally, **Technical Leads** provide architectural guidance, helping teams translate requirements into viable system designs. Together, these roles ensure a thorough, well-coordinated analysis phase that lays a strong foundation for development.

### KEY OUTPUTS

At the end of the analysis phase, several key outputs are produced to guide the next steps of system development. One major deliverable is the **F**unctional Requirements Document (FRD), which clearly lists all the functionalities the system must provide. This document serves as a blueprint for designers and developers, ensuring that nothing critical is overlooked. Use case scenarios are also created to describe specific situations in which users interact with the system. These scenarios help developers and testers understand how the system should behave in real-world situations. Lastly, process models are developed to illustrate the workflow and logic the system must support. These models give a clear, visual representation of business processes, making it easier for stakeholders to confirm that the requirements have been accurately captured. Together, these outputs form a solid foundation for moving confidently into the design phase.

## DESIGN



### MAIN FUNCTIONS

The design phase plays a crucial role in shaping the software development process by transforming abstract requirements into tangible technical plans. During this stage, teams convert the previously gathered software requirements into comprehensive design specifications that serve as blueprints for developers. This translation ensures all functional and non-functional requirements are properly interpreted and prepared for implementation.

A key focus of this phase involves designing intuitive user interfaces that prioritize usability and efficiency, creating seamless interactions for end-users. Simultaneously, developers establish the database architecture, carefully structuring data storage, relationships, and access methods to support the system's functionality while optimizing performance. These parallel design efforts - interface creation and database modeling - work together to form a cohesive technical foundation for the development team to build upon in subsequent phases.

### KEY INPUTS

The design phase produces essential artifacts that guide development. **Technical drawings** provide visual representations of the software architecture, including system components, data flows, and relationships, helping teams understand the structural framework. Additionally, **design documents** compile all critical design decisions, covering aspects like system behavior, interface layouts, database schemas, and technology choices. Together, these deliverables ensure clarity, consistency, and alignment across the development team before implementation begins.

### MAJOR TOOLS

The design phase relies on specialized tools to effectively translate requirements into actionable blueprints. **Figma** serves as a collaborative interface design platform, enabling teams to create and refine UI/UX prototypes with real-time feedback. For system modeling, **Unified Modeling Language (UML)** provides standardized visual diagrams - including class, use case, and sequence diagrams - that clearly represent software architecture and behavior before implementation begins. Project coordination is facilitated through **Trello's Kanban-style boards**, which help organize design tasks and track deliverables visually. **Version control systems** maintain design integrity by tracking file changes and enabling seamless collaboration across teams. For complex system design, **Modelio** offers comprehensive CASE tool functionality, supporting requirements management, visual modeling, and even code generation to bridge the gap between design and development. Together, these tools ensure a structured, efficient design process that aligns technical implementation with project requirements.

### MAJOR TECHNIQUES

The design phase transforms requirements into actionable technical plans through several critical processes. Prototyping helps visualize requirements by creating wireframes and basic interactive models that demonstrate user experience flows. The High-Level Design (HLD) establishes the system's structural framework, showing how major components will interact to meet functional goals. For deeper technical specification, Low-Level Design (LLD) details implementation aspects like data structures, algorithms, and module-specific logic. Architectural Design defines the system's foundational organization, including technology choices and component relationships. Concurrently, Database Design structures data storage solutions by selecting appropriate database systems, designing schemas, and optimizing data relationships for performance and scalability. Together, these activities create comprehensive technical blueprints that guide efficient development while ensuring alignment with both user needs and system requirements.

**SPECIALISTS**

The design phase brings together diverse expertise to create effective software solutions. **UX Designers** focus on crafting intuitive, user-centered interfaces that ensure logical and seamless interactions with the product. **Software Developers** contribute technical expertise, evaluating design feasibility and providing implementation insights to balance usability with technical constraints. **Business Analysts** bridge stakeholder needs and design execution by translating business requirements into clear guidelines for UX teams while reviewing designs for alignment with objectives. Overseeing the entire process, the **Project Manager** coordinates these efforts, making strategic decisions to keep the project on track while balancing user needs, technical realities, and business goals. Together, this cross-functional collaboration ensures designs are both user-friendly and technically executable while meeting project requirements.

### KEY OUTPUTS

The Design Document Specification serves as the comprehensive technical blueprint for development teams, containing all critical details needed to transform design concepts into functional code. This document systematically outlines the software architecture, component relationships, data structures, algorithms, interface specifications, and integration protocols. It provides developers with clear implementation guidelines by specifying technical requirements, system behaviors, and module interactions while establishing design patterns, coding standards, and technology constraints. By documenting these elements thoroughly, the specification ensures consistency across development teams, facilitates accurate effort estimation, and serves as a reference point throughout the coding phase to maintain alignment with the original design vision and project requirements. The document typically evolves through iterative reviews with stakeholders to address feasibility concerns before development commences.

## **IMPLEMATION**

### MAIN FUNCTION

The implementation phase represents the transformative stage where conceptual designs become operational software through systematic execution. Development teams write, integrate, and refine code based on the approved design specifications, translating architectural diagrams and prototypes into functional modules. This phase emphasizes rigorous **testing** at multiple levels - including unit, integration, and system tests - to verify functionality, performance, and security before deployment. Finally, the validated software undergoes **deployment**, where it is strategically released to production environments, configured for user access, and integrated with existing systems. These implementation activities ensure the delivered solution reliably meets both technical specifications and user requirements while maintaining system stability during rollout.

### KEY INPUTS

The implementation phase consists of two critical activities that bring software solutions to life. In the **Code and Development** stage, developers translate design specifications into executable software by writing functional code using appropriate programming languages, frameworks, and tools. This process focuses on creating clean, efficient, and maintainable code that precisely follows the technical blueprints while meeting performance requirements. Following successful development, **System Deployment** prepares the software for real-world use through a structured rollout process. Deployment typically begins in staging environments for final validation before progressing to live production systems, ensuring smooth transition to end-users. This phase includes crucial steps like environment configuration, database setup, and system integration to deliver a fully operational product that aligns with both technical specifications and user expectations. Together, these implementation activities transform theoretical designs into practical, accessible software solutions.

### MAJOR TOOLS

Several powerful tools streamline the implementation and quality assurance process in software development. Selenium provides a comprehensive suite for automating web browser testing, enabling efficient validation of functionality across different browsers and platforms. For continuous integration and delivery, Jenkins automates critical workflows including building, testing, and deploying code changes. Code review platforms like GitHub, GitLab, and SonarQube facilitate collaborative code inspection, helping teams maintain quality standards through peer reviews and automated analysis. To manage defects, bug tracking systems such as Jira and Bugzilla offer specialized features to log, prioritize, and monitor issues throughout the development lifecycle. Together, these tools enhance code quality, accelerate delivery, and ensure robust software through systematic testing and issue management.

### MAJOR TECHNIQUES

The implementation process involves several key practices to ensure successful software deployment and adoption. **Continuous Testing and Feedback** plays a crucial role, where teams regularly test the product and gather user feedback to verify it meets changing requirements. The **Phase Development** approach helps mitigate risks by gradually releasing the software to select users or environments before expanding to the full user base. To facilitate smooth transitions**, User Training** educates end-users on operating the new system and adapting it to their daily workflows. After deployment, **Post-Implementation Support** provides ongoing assistance to address any emerging issues or user questions. Throughout the process, **Effective Communication** maintains clear channels between development, testing, and operations teams to ensure alignment. Additionally, enforcing \*\*Code Quality Standards\*\* through consistent coding practices, thorough reviews, and automated tools helps maintain high-quality, sustainable code. Together, these practices create a robust framework for successful software implementation.

### SPECIALISTS

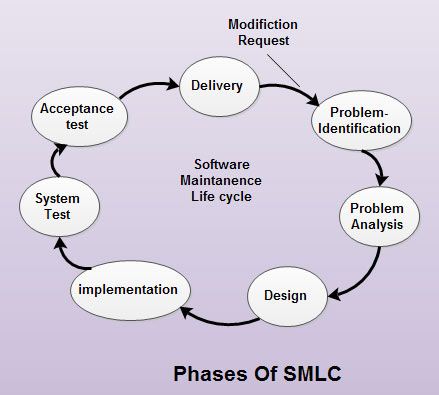
The success of the implementation phase relies on the collaboration of key team members, each playing a distinct role. **Software engineers** form the backbone of this phase, taking primary responsibility for writing, refining, and debugging the code that brings the software to life. Working alongside them, **testers** perform comprehensive quality checks through rigorous testing protocols to identify and resolve bugs, ensuring the final product meets both technical standards and user needs. Overseeing these technical activities, the **project manager** maintains strategic control of the implementation, coordinating team efforts while carefully monitoring timelines and budgets to keep the project on track. Complementing these roles, **system analysts** contribute their technical expertise to verify that all implementation work remains aligned with the predefined system requirements and architectural vision. Together, this cross-functional team ensures the software evolves from design specifications to a fully functional, high-quality product ready for deployment.

### KEY OUTPUTS

The implementation phase culminates in two critical outcomes: First, a thoroughly tested version of the product that has undergone rigorous quality checks and is deemed stable for deployment in a production environment. This version has passed all test cases, meets performance benchmarks, and addresses identified bugs.

Second, the release of a fully functional product to end-users, marking the completion of development efforts. This deliverable represents a complete software solution that aligns with all specified requirements and is ready for real-world use, complete with necessary documentation and support systems. Together, these deliverables signify the successful transition from development to operational software.

## **MAINTENANCE**



### MAIN FUNCTIONS

The maintenance phase plays a crucial role in the software lifecycle by focusing on three key objectives: safeguarding system integrity, optimizing performance, and implementing strategic enhancements. This critical stage ensures the software remains fully operational through regular updates and prompt resolution of critical issues while proactively addressing security vulnerabilities to protect against emerging threats. Maintenance teams continuously monitor and fine-tune system performance to deliver optimal user experiences, and when necessary, introduce carefully planned new features to keep the software aligned with evolving user needs and technological advancements. By maintaining this balance between stability and innovation, organizations can extend their software's lifespan, maximize return on investment, and ensure the solution remains secure, efficient, and valuable to users over time.

### KEY INPUTS

The maintenance phase involves crucial processes to keep software effective and efficient. **Enhancement** focuses on adapting the software to evolving user and business needs by adding new features or functionality, ensuring the system remains valuable and competitive over time. Simultaneously, **performance optimization** involves continuous monitoring and refinement of the software's speed, resource usage, and responsiveness. Maintenance teams analyze system metrics, identify bottlenecks, and implement improvements to ensure the application runs efficiently while consistently meeting its performance benchmarks. These ongoing efforts help extend the software's lifespan while maintaining optimal operation in changing technical environments.

### MAJOR TOOLS

To effectively maintain software systems, developers rely on specialized tools that streamline updates and ensure stability. **Program slicers** enable precise code modifications by isolating specific sections that require refactoring while preserving overall functionality. **Static analyzers** automatically scan codebases to detect syntax errors, coding standard violations, and potential quality issues that manual reviews might overlook. For runtime-specific problems, **dynamic analyzers** identify execution-related vulnerabilities like memory leaks, race conditions, and runtime crashes that only manifest during operation. **Transformation tools** assist in systematically restructuring code to improve maintainability without altering external behavior. Complementing these, **dependency analyzers** map and monitor component relationships to maintain system integrity when implementing changes for bug fixes or new features. Together, these tools form a comprehensive maintenance toolkit that helps developers enhance software while minimizing risks to system reliability throughout its lifecycle

### Major Maintenance Techniques

The maintenance phase employs several critical techniques to ensure software remains robust and effective. **Corrective maintenance** tackles post-deployment issues by systematically identifying, diagnosing, and resolving bugs, performance bottlenecks, and security vulnerabilities through careful analysis and patching. **Continuous monitoring** providesreal-time oversight of system health, tracking performance metrics, stability indicators, and security alerts to proactively detect emerging concerns.

For code quality improvement, **code analysis and refactoring** techniques help optimize existing implementations by evaluating structural integrity, eliminating technical debt, and enhancing readability without altering functionality. **Configuration management** maintains version control across environments, ensuring proper deployment synchronization and change traceability throughout the software's lifecycle.

Finally, **perfective maintenance** enhances system capabilities by implementing user-requested features or performance upgrades that align with evolving business needs. Together, these techniques form a comprehensive maintenance strategy that preserves software reliability while enabling controlled evolution.

### Key Maintenance Specialists and Roles

Several critical roles collaborate to maintain software systems effectively. **Developers** serve as the primary technical resource, addressing bugs, implementing updates, and adding new features to enhance functionality. **System managers** oversee operational health by continuously monitoring performance metrics and making necessary adjustments to maintain optimal system behavior.

**Customers play** a vital role as end-users by reporting issues, providing valuable feedback on system performance, and often participating in acceptance testing for major updates. For complex architectural changes, **technical architects** contribute their expertise by evaluating system impacts and designing modification strategies that preserve structural integrity while implementing improvements.

Together, these specialists form a maintenance ecosystem that combines technical execution with user feedback and architectural oversight to keep software systems reliable, efficient, and aligned with organizational needs.

### Key Maintenance Outputs

The maintenance phase produces two essential outcomes that contribute to long-term software success. First, it delivers \*\*a fully monitored system\*\* equipped with ongoing evaluation mechanisms that track performance, security, and usability - creating a foundation for continuous improvement cycles. Second, it \*\*ensures sustained alignment\*\* between the software's capabilities and evolving user requirements through regular updates and optimizations. Together, these outputs maintain the software's relevance, reliability, and performance throughout its lifecycle while adapting to changing technological landscapes and business needs.

REFERENCES

<https://www.linkedin.com/planning-phase-in-sdlc>

<https://www.superworks.com>

<https://www.aicura.com>

<https://www.google.com>

<https://www.itexus.com/analysis-phase>

<https://itexus.com/implementation-phase>

<https://www.ecomstreet.com>

<https://GeeksforGeeks.org>

<https://teachingagile.com/maintenance-phase>

<https://www.idapgroup.com/design-phase>

<https://kms-solutions.asia>

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Phase | Main  Function | Key  inputs | Major  Tools | Major  Techniques | Key  Specialists And  Task | Key  Outputs |
| Planning | feasibility studies are conducted,  defining project scope and objectives | 1.Budget Constraints  2.tool selection | 1. Gantt Charts, 2.**Kanban boards**,  3.**task lists** , **4.project roadmaps** ,  5 **tree diagrams** | 1.Operational planning  2.Contingency planning.  3.Critical path analysis.  4.SWOT analysis. | Project Manager.  2. Business 3.Analysts.Stakeholder.  4Developers | Timeline & Milestones  2.Project Charter |
| Analysis | 1.requirements documentation  2. feasibility analysis | 1.documentation of existing systems  2.observation of current workflows. | 1. **Dataflow diagrams**  2.  **Software Requirements Specifications**  **3. Use cases**  4. **user stories** | 1. Interviews  2. Prototyping  3.Questionnaires  4.Observation  5.Workshop | 1.Project Manager  2. Business Analysts  3.System Analysts  4.Technical Lead | 1.Process Models  2. Functional Requirements Document (FRD), |
| Design | 1.transforming abstract requirements  2. designing intuitive user interfaces | 1. Technical drawings  2. design documents | 1. Unified Modeling Language (UML)  2.Figma  3.Trello's Kanban-style boards,  4.Version control systems  5. Modelio | 1. Prototyping  2.High-Level Design (HLD)  3. Low-Level Design (LLD)  4. Architectural Design | 1. UX Designers  2.Software Developers  3. Business Analysts  4. Project Manager | 1. Design Document Specification |
| Implementation | 1.testing  2. deployment | 1. Code and Development  2. System Deployment | 1. Selenium  2. Jenkins  3. Bugzilla  4. SonarQube  5. Jira | 1. Continuous Testing and Feedback  2. Phase Development  3. User Training  4. Post-Implementation Support  5. , Effective Communication | 1. Software engineers  2.Project Manager  3. Business Analysts  4.  Testers | 1. Software engineers  2. a fully functional product to end-users |
| Maintenance | 1.ensures long-term performance  2. security, and relevance through updates | 1. performance optimization  2. Enhancement | 1.Program slicers  2.Static analyzers  3. dynamic analyzers  4. Transformation tools  5.dependency analyzers | 1. Corrective maintenance  2. Continuous monitoring  3. Configuration management  4. perfective maintenance  5. code analysis and refactoring | 1. Developers  2. System managers  3. Customers  4. technical architects | 1. fully monitored system  2. ensures sustained alignment |

**Conclusion**

The Systems Development Life Cycle (SDLC) provides a structured and systematic approach to developing high-quality, efficient, and sustainable software solutions. By following its well-defined phases—**Planning, Analysis, Design, Implementation, and Maintenance**—organizations can ensure that their systems meet business objectives, align with user needs, and adapt to evolving technological demands.

Each phase of the SDLC plays a critical role in the development process:

* **Planning** establishes a clear roadmap, ensuring feasibility and alignment with business goals.
* **Analysis** defines precise requirements and validates system viability before development begins.
* **Design** transforms requirements into technical blueprints, creating a foundation for development.
* **Implementation** brings the system to life through coding, testing, and deployment.
* **Maintenance** ensures long-term performance, security, and relevance through updates and optimizations.

By leveraging appropriate **tools, techniques, and specialist expertise** at each stage, development teams can minimize risks, enhance efficiency, and deliver robust software solutions. The SDLC not only facilitates structured development but also ensures that systems remain scalable, secure, and aligned with organizational needs over time.

Ultimately, the SDLC serves as an indispensable framework for successful software development, enabling businesses to innovate, optimize operations, and maintain a competitive edge in an increasingly digital world.

Contents

[**Introduction** 1](#_Toc197065239)

[GENERIC PHASES OF SYSTEM DEVELOPMENT LIFE CYCLE 2](#_Toc197065240)

[**1.** **PLANNING** 2](#_Toc197065241)

[Main Functions 3](#_Toc197065242)

[KEY INPUTS 3](#_Toc197065243)

[**MAJOR TOOLS** 4](#_Toc197065244)

[Major Techniques 4](#_Toc197065245)

[Specialists 4](#_Toc197065246)

[Key outputs 5](#_Toc197065247)

[**1.** **ANALYSING** 5](#_Toc197065248)

[Main Functions 6](#_Toc197065249)

[KEY INPUTS 6](#_Toc197065250)

[**MAJOR TOOLS** 7](#_Toc197065251)

[MAJOR TECHNIQUES 7](#_Toc197065252)

[SPECIALISTS 8](#_Toc197065253)

[KEY OUTPUTS 8](#_Toc197065254)

[2. DESIGN 9](#_Toc197065255)

[MAIN FUNCTIONS 9](#_Toc197065256)

[KEY INPUTS 9](#_Toc197065257)

[MAJOR TOOLS 10](#_Toc197065258)

[MAJOR TECHNIQUES 10](#_Toc197065259)

[KEY OUTPUTS 11](#_Toc197065260)

[**3.** **IMPLEMATION** 12](#_Toc197065261)

[MAIN FUNCTION 12](#_Toc197065262)

[KEY INPUTS 12](#_Toc197065263)

[MAJOR TOOLS 13](#_Toc197065264)

[MAJOR TECHNIQUES 13](#_Toc197065265)

[SPECIALISTS 14](#_Toc197065266)

[KEY OUTPUTS 15](#_Toc197065267)

[**4.** **MAINTENANCE** 15](#_Toc197065268)

[MAIN FUNCTIONS 16](#_Toc197065269)

[KEY INPUTS 16](#_Toc197065270)

[MAJOR TOOLS 16](#_Toc197065271)

[Major Maintenance Techniques 17](#_Toc197065272)

[Key Maintenance Specialists and Roles 18](#_Toc197065273)

[Key Maintenance Outputs 18](#_Toc197065274)