MERiT

System Development Life Cycle (SDLC)

An Overview

Introduction to SDLC

Definition

- SDLC = **S**ystem **D**evelopment **L**ife **C**ycle
- It is a **systematic approach** for developing software.
- It emerged from the need to **handle the growing complexity of software** in the 1960s and 1970s.
- The **S** in **SDLC** stands for **system**, not **software**, because SDLC addresses the entire system, including software, hardware, people, processes, data, and more.

Key Objectives

- **Deliver systems** that meet expectations.
- **Stay within budget and time constraints** through planning and phased execution.
- **Ensure alignment with business objectives** so that the solution delivers real value.
- **Manage complexity and uncertainty** by breaking down the lifecycle into manageable parts.

Key Objectives

- **Enable maintainability and evolution** through clear documentation and modularity.
- **Facilitate communication** among technical and non-technical stakeholders.
- **Support risk management** by making issues visible early and often.

History

- **Pre-1960s:** The Birth of Software
- **1960s:** The Rise of Complexity and the Software Crisis
- **1970s:** Waterfall Model and Formalization
- **1980s–1990s:** Iteration, Risk, Modeling
- **2000s:** Agile Manifesto
- **2010s:** Automation (CI/CD) and DevOps
- **2020s:** Hyperautomation

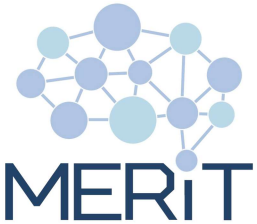
SDLC Phases

Definition

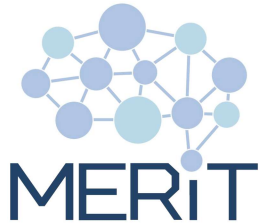
- A phase represents a **distinct stage** in the overall SDLC process, focusing on **specific activities** and **deliverables that lay the groundwork for the next phase**.
- This approach provides a **conceptual roadmap** from idea to software system, treating it as a **living organism**.
- The SDLC is commonly divided into 7 high-level phases.

Phases

- Planning
- Analysis
- Design
- Implementation
- Testing
- Deployment
- Maintenance



Phases



- Planning: *Should this system **be developed?***
- Analysis: ***What** should the system **do?***
- Design: ***How** should the system **work?***
- Implementation: ***Build** the system.*
- Testing: *Does the system **work as intended?***
- Deployment: ***Release** the system to users.*
- Maintenance: ***Keep** the system **running and up to date**.*

Planning

- The planning phase is the **initial stage** of the SDLC where the **feasibility** is evaluated to **authorize further development**.
- Mistakes in this phase can result in project misalignment, unrealistic expectations, and eventual failure in execution.

Planning – Key Activities

- Define the **problem** and **objectives**.
- Identify and engage **stakeholders**.
- Outline **scope**, **constraints**, and **assumptions**.
- Assess **feasibility** from technical, economic, operational, legal, and scheduling angles.
- Draft **high-level roadmap** outlining resources, timeline, milestones, and deliverables.

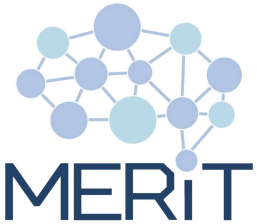
Planning – Key Deliverables

- Business Case
- Stakeholder Register
- Preliminary Scope Statement
- Feasibility Analysis Report
- Initial Risk and Assumption Log
- High-Level Project Plan

Analysis

- The analysis phase involves **defining and validating requirements** from both non-technical and technical perspectives to **outline the system's expected behavior**.
- Mistakes in this phase can result in building the wrong system, scope creep, and costly rework in later stages.

Analysis – Key Activities

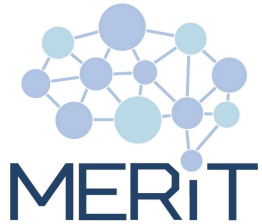


- Elicit **functional and non-functional requirements** from users and stakeholders.
- **Model processes**.
- **Validate** and **refine requirements** with stakeholders.
- **Prioritize requirements** against value, risk, and dependency.
- **Establish traceability** between requirements and future design/tests.

Analysis – Key Deliverables

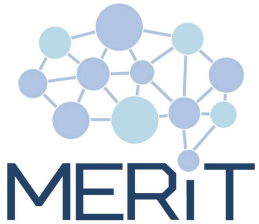
- System Requirements Specification (SRS)
- Process and Data Models (use-case, data-flow, etc.)
- Requirements Validation Report
- Requirements Prioritization Matrix
- Requirements Traceability Matrix

Design



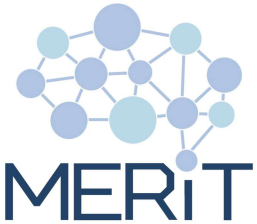
- The design phase involves **translating the requirements** into both high-level and low-level **designs** that will **guide the implementation**.
- Mistakes in this phase can result in inefficient, inflexible, and unscalable systems that are difficult to implement and maintain.

Design – Key Activities



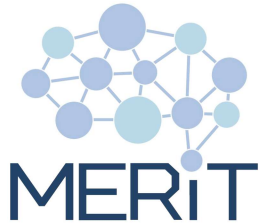
- Transform requirements into **system architecture**.
- Select **technology stack**.
- Evaluate **alternatives** and **proof of concepts**.
- Design **system interactions** and **data structures**.
- Design **user interface** and **interactions**.
- Address **system-wide concerns** (security, performance, scalability, etc.).
- **Review** and baseline the design with key stakeholders.

Design – Key Deliverables



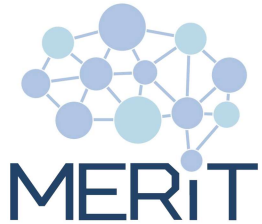
- Architectural Design Document (high-level design)
- Component and Interface Specifications (low-level design)
- Data Models and Schema Designs
- API and Message Contracts
- UI Wireframes and Prototypes
- Updated Traceability Matrix

Implementation



- The implementation phase involves **executing the design through coding** that collectively fulfills the **system's intended functionality**.
- Mistakes in this phase can result in defects and increase technical debt.

Implementation – Key Activities



- Break down the design into implementation **tasks**.
- Write **code** that fulfils functional requirements.
- Develop **unit and integration tests**.
- Conduct peer **code reviews**.
- Maintain **traceability** between requirements, design, and code.

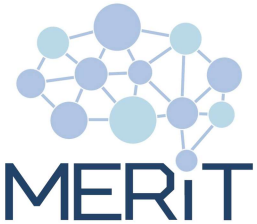
Implementation – Key Deliverables

- Source Code
- Unit and Integration Test Suites
- Code Review Records
- Release Candidate
- Technical Documentation

Testing

- The testing phase involves **evaluating the system** to ensure it **operates as intended**.
- Mistakes in this phase can result in system failures and reputational damage.

Testing – Key Activities



- Verify system **correctness** and **quality**.
- Prepare test **strategy**.
- Write **test cases**.
- Set up related **test environments** and **test data**.
- **Execute** tests (unit, integration, system, acceptance, etc.).
- Log and track **defects**; verify **fixes**.

Testing – Key Deliverables

- Test Plan
- Test Cases
- Test Execution Logs and Defect Reports
- Test Summary Report
- Updated Traceability Matrix

Deployment

- The deployment phase involves **releasing the system** into the **target environment** for use by its **intended users**.
- Mistakes in this phase can result in downtime, data loss, and disruptions to business operations.

Deployment – Key Activities

- **Package** and configure the system for the target environments.
- **Coordinate** cut-over activities and stakeholder communications.
- Document **release notes** and **user instructions**.
- Confirm **successful rollout** or trigger **fallback procedures** if needed.

Deployment – Key Deliverables

- Deployment Checklist and Execution Record
- Configured Environment and Release Package
- Release Notes and User/Support Guides
- Deployment Verification Report
- Rollback / Fallback Plan

Maintenance

- The maintenance phase is an **ongoing process** that supports, corrects, and enhances the system **throughout its lifespan**.
- Mistakes in this phase can result in degraded system performance, increased vulnerability, and a shorter system lifespan.

Maintenance – Key Activities

- **Monitor** system performance, availability, and user feedback.
- **Diagnose** and **correct defects**; issue patches and minor enhancements.
- **Adapt** the system to evolving business rules or platform changes.
- **Maintain** documentation, support knowledge base, and training materials.
- **Plan** for major revisions or controlled end-of-life transition.

Maintenance – Key Deliverables

- Change & Patch Logs
- Updated User and Support Documentation
- Operational Metrics and Service Reports
- Enhancement Release Packages
- Backup and Recovery Procedures

SDLC Models

Definition

- A model is a **conceptual framework** that defines the **sequence**, **flow**, and **management style** of the **SDLC phases**.

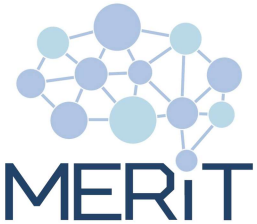
Evolution

- Models have **evolved over time** to address complexity, uncertainty, risks, and speed.
- Models have shifted **from rigid,** plan-driven approaches **to flexible**, iterative methods.
- **Projects differ** in their requirements, resources, risks, volatility, compliance, and urgency.
- Mastering various models enables building **customized model combinations** that provide optimal value tailored to the specific needs of the project.

Early Models

- 1970s
- e.g., Waterfall
- 🚀 They were borrowed from the construction industry.
- 🧭 They follow a linear, phase-by-phase approach prioritizing thorough planning.
- 😩 They struggle to handle uncertainty and change.

Iterative and Incremental Models

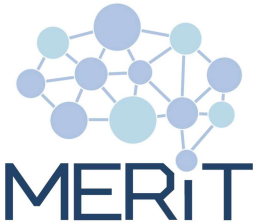


- 1980s – 1990s
- e.g., Iterative, Incremental, Spiral
- 🚀 Projects were failing due to rigid plans. Clients were unable to fully specify requirements upfront.
- 🧭 They focus on building incrementally expanding systems through an iterative process.
- 🙂 They integrated feedback and refinement (even risk assessment in Spiral) into each cycle.

Agile Revolution

- 2000s
- e.g., Scrum
- 🚀 Software complexity grew, business environments changed rapidly, and users needed to see and use software quickly.
- 🧭 They focus on iterative cycles, fast delivery, customer collaboration, and flexible change handling.
- 🙂 They addressed long delays and poor adaptability and integrated customer feedback.

DevOps Movement

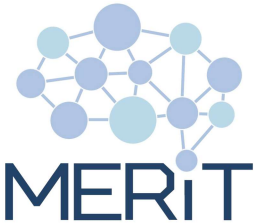


- 2010s
- e.g., DevOps, Agile + UX, SAFe
- 🚀 The need to go beyond development, focusing on automation, scalability, deployment, and continuous improvement, has risen.
- 🧭 They build unified teams of development, operations, testing, and even business (BizDevOps).
- 🙂 They enabled CI/CD pipelines, real-time monitoring, and cloud-native practices.

Classification

- Plan-Driven Models: Waterfall, V, Spiral
- Adaptive Models: Iterative and Incremental and, Agile, Rapid Application Development (RAD)
- Continuous and Integrated Models: DevOps, Site Reliability Engineering (SRE), BizDevOps
- Hybrid and Contextual Models: Agile-Waterfall, Disciplined Agile Delivery (DAD), Scaled Agile Framework (SAFe)

Key Characteristics for Comparison



- Flexibility and Change Tolerance: How easily allows changes once development has started.
- Involvement: Degree to which users and stakeholders engage in the development process.
- Delivery: Whether the product is developed in cycles, allowing for partial deliveries and early feedback.
- Risk Management: How explicitly and early identifies and manages risks.

Key Characteristics for Comparison

- Time to Market: How quickly a working version is delivered to users.
- Documentation Rigor: Extent and formality of documentation required.
- Team Structure and Discipline: The type of team dynamics each model supports or requires.

Key Roles in SDLC

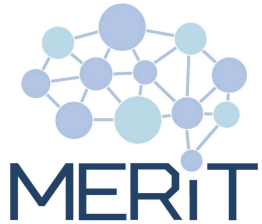
Project Manager

- Oversees the entire project planning.
- Interfaces between technical and non-technical stakeholders.

Business Analyst

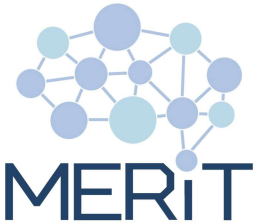
- Gathers and documents system requirements.
- Acts as a bridge between business stakeholders and technical teams.

Software Architect



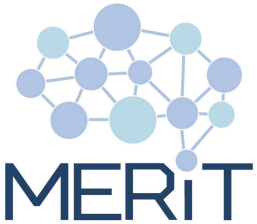
- Designs the high-level architecture of the system.
- Makes critical technology and design decisions.

Developer / Software Engineer



- Implements the system through coding, integration, and unit testing.
- Participates in reviews and supports the resolution of issues.

Tester / QA Engineer

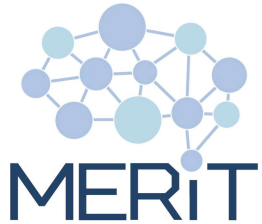


- Designs and executes test cases.
- Verifies that the system meets quality standards and requirements.

UI/UX Designer

- Designs user interfaces and user experiences.
- Ensures usability and accessibility standards are met.

DevOps Engineer

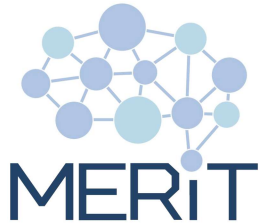


- Manages infrastructure, deployment, and automation.
- Ensures system reliability and delivery pipelines.

Product Owner

- Provides business priorities, makes decisions on scope and features.
- Accepts or rejects deliverables.

Non-Technical Stakeholders



- Influence requirements, funding, and acceptance.
- Provide feedback and business context.