THE COMPLETE SOFTWARE DEVELOPMENT PROCESS

A practical guide to project documentation, requirements, design, development, testing, deployment and maintenance

Prepared as a practical reference for software owners, managers, analysts, developers and users.
Version 1.0 | 21 October 2026

Contents

- 1. Introduction
- 2. Complete Software Development Lifecycle
- 3. Documentation Deliverables in Order
- 4. What an SRS Must Contain
- 5. Functional and Non-Functional Requirements
- 6. Approval Gates and Sign-Off Process
- 7. Requirements Traceability Matrix
- 8. Supporting Project Registers
- 9. Recommended Documentation Folder Structure
- 10. Roles and Responsibilities
- 11. Example: Applying the Process to a PACS Probation Module
- 12. Key Rules for Successful Software Delivery
- 13. Initial Project Checklist

1. Introduction

Developing good software is not just writing code. It is a structured process that begins with identifying a problem, documenting requirements, designing the solution, developing the system, testing it, deploying it and maintaining it.

Core principle: Document what the system must do before deciding how to build it, and agree on the design before writing the code.

Proper documentation ensures that the developer, client, management, testers and end users understand what is being built, how it should work, how it will be tested and what must be delivered.

2. Complete Software Development Lifecycle

1. **Project initiation and feasibility study —** Establish the problem, objectives, scope, budget, risks and viability.
2. **Requirements gathering and analysis —** Understand users, business processes, needs and constraints.
3. **Software Requirements Specification (SRS) —** Document exactly what the software must do.
4. **System analysis and process modelling —** Map workflows, use cases, business rules and data movement.
5. **Software architecture and system design —** Define components, technology, database, interfaces and security.
6. **User interface and user experience design —** Prepare wireframes, screen designs and prototypes.
7. **Implementation and coding —** Develop the frontend, backend, database and integrations.
8. **Testing and quality assurance —** Verify the system against the agreed requirements.
9. **User acceptance testing (UAT) —** Users confirm that the software meets their actual needs.
10. **Deployment and release —** Install, configure, secure and release the system into production.
11. **Training and handover —** Provide manuals, training, source code and technical documentation.
12. **Maintenance and continuous improvement —** Fix defects, improve features, update dependencies and monitor performance.

These stages are not always strictly one-way. In Agile development, they repeat for each development iteration. However, required documentation, approvals and quality checks should still be maintained.

3. Documentation Deliverables in Order

Phase A — Project initiation

1. Project Proposal

- Background and problem statement.
- Proposed solution and project objectives.
- Expected benefits and target users.
- Estimated cost, duration and resources.
- Initial risks and recommendations.

Approval: project sponsor, client or management.

2. Feasibility Study

- Technical feasibility: whether the technology can support the system.
- Economic feasibility: whether the benefits justify the costs.
- Operational feasibility: whether the organization can use and support it.

- Legal feasibility: compliance with applicable laws and policies.
- Schedule feasibility: whether the project can be delivered in the available time.
- Security and privacy risks.

Output: feasibility report and go/no-go recommendation.

3. Project Charter and Scope Statement

- Project owner, project manager, objectives and success criteria.
- Deliverables and included/excluded features.
- Budget, schedule, milestones and responsibilities.
- Approval authority and governance arrangements.

Purpose: formally authorize and define the project.

4. Stakeholder and User Analysis

- Identify all people and groups affected by the system.
- Record their responsibilities, decisions, information needs and expectations.

Output: stakeholder register, user categories and initial responsibility matrix.

Phase B — Requirements gathering and analysis

5. Requirements Elicitation Report

- Record interviews, workshops, observation, questionnaires and review of existing documents.
- Document current processes, problems, user expectations, policies and unresolved questions.

Output: findings and validated needs.

6. Business Requirements Document (BRD)

- Business objectives and operational problems.
- High-level business requirements, stakeholders and constraints.
- Success measures and scope.

Difference from SRS: the BRD explains the business need; the SRS specifies the required software behaviour.

7. Software Requirements Specification (SRS)

- Document clear, uniquely numbered and testable functional and non-functional requirements.
- Define roles, business rules, data, interfaces, reporting, exceptions and acceptance criteria.

This is a central agreement between the organization and the development team.

8. System Analysis and Process Models

- Use-case diagrams and detailed use-case descriptions.
- Current-state and proposed workflows.
- Activity diagrams and, where useful, data-flow diagrams.
- Business rules, exception scenarios and initial requirements traceability matrix.

Purpose: explain how users and the system interact.

Phase C — System design

9. Software Architecture Document (SAD)

- Major components and their responsibilities.
- Communication, deployment model and technology choices.
- Architecture diagrams and key technical decisions.

Examples include layered architecture, client-server, modular monolith or microservices.

10. Database Design Document

- Entities, tables or collections, fields and data types.
- Keys, relationships, constraints, indexes and validation rules.
- Data retention, migration, backup and recovery strategy.

Output: entity-relationship diagram (ERD), data dictionary and schema.

11. Security and Access Control Design

- Authentication, roles, permissions and data visibility.
- Audit logs, encryption, session management and monitoring.
- Threat mitigation and role-permission matrix.

Define who may view, create, change, approve, export or delete each category of data.

12. User Interface and User Experience (UI/UX) Design

- User journeys, wireframes and clickable prototypes.
- Screen layouts, navigation, forms, dashboards, tables and reports.
- Validation messages, accessibility and responsive design.

Users and authorized business representatives should review key screens before full implementation.

13. API and Integration Specification

- Endpoint names and HTTP methods.
- Request and response formats, authentication and permission rules.
- Validation errors, versioning and third-party integrations.

Defines communication between frontend, backend and external services.

14. Detailed Technical Design / Low-Level Design (LLD)

- Module responsibilities and class/function designs.
- Sequence diagrams, state transitions, validation logic and algorithms.
- Error handling and component-level implementation details.

Architecture is the big picture; low-level design explains individual components.

Phase D — Development, testing and release

15. Development Plan and Coding Standards

- Modules, tasks, assigned developers and milestones.
- Branch strategy, naming conventions and code review rules.
- Environment setup and configuration management.

Purpose: make development consistent and traceable.

16. Software Test Plan and Test Cases

- Testing scope, responsibilities, environments and entry/exit criteria.
- Test scenarios mapped to SRS requirements.
- Expected results and evidence to collect.

Include functional, integration, security, performance and usability testing as appropriate.

17. Test Reports and Defect Register

- Executed tests and expected versus actual results.
- Pass/fail status, severity, screenshots or logs.
- Assigned owner, fix details and retest results.

Purpose: show what was tested and what remains unresolved.

18. User Acceptance Testing (UAT) Plan and Sign-off

- Realistic business scenarios executed by representative users.
- Evidence, outstanding defects and accepted limitations.
- Formal acceptance or rejection by the authorized representative.

UAT confirms whether the system meets agreed business needs.

19. Deployment and Release Plan

- Production setup and environment configuration.
- Database migration, backups, release steps and rollback procedures.
- Monitoring, release approval and release notes.

Purpose: release safely and recover if deployment fails.

20. User Manual, Technical Manual and Training Materials

- User instructions and role-specific guides.
- Installation, configuration, maintenance and troubleshooting instructions.
- Training materials and support contacts.

Keep user guidance separate from technical maintenance instructions where practical.

21. Handover and Maintenance Documentation

- Source code and repository access, as agreed.
- Deployment instructions, backup and restoration procedures.
- Environment-variable documentation without real secrets.
- Licences, support arrangements, known defects and maintenance responsibilities.

Purpose: enable the organization to operate and maintain the software.

4. What an SRS Must Contain

A common reference for structuring requirements is ISO/IEC/IEEE 29148. Templates vary by organization and project, but a useful SRS should cover the following sections.

SRS section	Contents
1. Introduction	Purpose, scope, intended audience and definitions.
2. Overall description	Product context, users, assumptions and constraints.
3. User roles	Each role and its responsibilities.

4. Functional requirements	Every required system function.
5. Business rules	Policies, calculations, conditions and restrictions.
6. Data requirements	Information stored, validated, retained and reported.
7. External interfaces	User interfaces, APIs and integrations.
8. Non-functional requirements	Security, performance, usability, availability and scalability.
9. Reports and analytics	Required reports, filters, calculations and exports.
10. Error and exception handling	What happens when something fails.
11. Acceptance criteria	How each requirement will be verified.
12. Appendices	Diagrams, glossary, references and supporting material.

Example of a properly written requirement

FR-AUTH-001 — User authentication

Requirement: The system shall authenticate a user using registered credentials before granting access to protected functions.

Precondition: The user has an active account.

Expected result: Valid credentials establish an authenticated session. Invalid credentials are rejected without revealing whether the username or password was incorrect.

Access control: The system shall enforce the user's assigned permissions on every protected operation.

Acceptance test: Test valid credentials, invalid credentials, a disabled account and a user without permission to access a protected function. Verify the expected result for each case.

Avoid vague requirements such as “the login should be good and secure.” Requirements should be clear, measurable where possible and testable.

5. Functional and Non-Functional Requirements

Functional requirement: What the system must do. Example: an authorized officer can create and update a case record.

Non-functional requirement: How well or under what constraints the system must operate. Example: confidential case information is access-controlled and security-relevant actions are recorded in an audit log.

Make non-functional requirements measurable wherever practical. Avoid statements such as “the system must be very fast” without defining a performance target and test conditions.

6. Approval Gates and Sign-Off Process

Gate 1: Project approval

Approve proposal, feasibility study, charter and initial scope before committing significant development resources.

Gate 2: Requirements approval

Review BRD and SRS with users, management and the developer. Resolve disagreements and approve a requirements baseline.

Gate 3: Design approval

Approve architecture, role permissions, database design, workflows, API specifications and key screen prototypes before implementing the relevant features.

Gate 4: Development review

Review working features against requirements, code quality, security controls and milestones. Record incomplete work and defects.

Gate 5: Testing and acceptance

Complete tests, resolve critical defects, conduct UAT and obtain formal acceptance from the authorized representative.

Gate 6: Production and handover

Authorize release, verify backups and recovery, deliver documentation and source code as agreed, and establish support arrangements.

If an approved requirement needs to change, document the change, assess its cost, schedule, security and design impact, and obtain approval from the designated authority. Do not silently change the agreed scope.

7. Requirements Traceability Matrix

The requirements traceability matrix connects each requirement to its design, implementation and test evidence. It helps show whether every promised feature was delivered.

Requirement ID	Requirement	Design reference	Test ID	Status
FR-AUTH-001	Authenticate users	AUTH-DES-01	TC-AUTH-001	To be verified
FR-USER-002	Assign roles and permissions	SEC-DES-02	TC-SEC-002	To be verified
FR-CASE-003	Register a case	CASE-DES-01	TC-CASE-003	To be verified
FR-REPORT-004	Generate monthly report	REP-DES-01	TC-REP-004	To be verified

These IDs are illustrative examples, not claims about a completed system.

The matrix should eventually show whether each requirement is approved, implemented, tested and accepted, and identify requirements that lack design, code or test evidence.

8. Supporting Project Registers

Register	Purpose
Project schedule and milestone tracker	Tracks planned versus actual progress.
Meeting minutes and decision register	Records decisions, responsibilities and deadlines.
Risk and issue register	Tracks risks, blockers and unresolved problems.
Change request register	Records requested and approved changes.
Defect and bug register	Tracks problems from discovery to resolution.
Configuration and environment register	Documents development, test and production environments.
Requirements traceability matrix	Links requirements to implementation and testing.
Version and release history	Shows what changed in each release.
Technical debt register	Records known technical weaknesses requiring later attention.
Maintenance and incident log	Records production problems, fixes and preventive actions.

Each document or register should have an owner, version number, date, status and approval history where appropriate.

9. Recommended Documentation Folder Structure

- **01_Project_Initiation/** Proposal, feasibility study, charter and scope.
- **02_Requirements/** Stakeholders, elicitation report, BRD and SRS.
- **03_System_Analysis/** Use cases, workflows, business rules and process diagrams.

- **04_System_Design/** Architecture, database, security, UI/UX, API and low-level design.
- **05_Project_Management/** Schedule, minutes, risks, issues and change requests.
- **06_Development/** Coding standards, technical notes and repository instructions.
- **07_Testing/** Test plan, test cases, test results and defect register.
- **08_UAT/** User test scripts, findings and acceptance sign-off.
- **09_Deployment/** Deployment plan, configuration, release notes and rollback.
- **10_User_Documentation/** User manual, role guides and training materials.
- **11_Handover_Maintenance/** Source code handover, support, backups and maintenance.

Keep approved documents under version control. Do not silently overwrite an approved SRS. Issue a new revision and record what changed, why, who approved it and when.

10. Roles and Responsibilities

Role	Main responsibility
Project sponsor or management	Approves business case, funding, scope and major decisions.
Business analyst	Gathers requirements and prepares the BRD, SRS and process models.
System architect	Prepares architecture and major technical decisions.
UI/UX designer	Prepares user journeys, wireframes and prototypes.
Database designer or engineer	Designs the data model and database implementation.
Software developer	Implements approved requirements and maintains technical documentation.
QA engineer or tester	Prepares and executes tests and reports defects.
Business users	Validate workflows, perform UAT and confirm usability.
Project manager	Coordinates schedule, risks, decisions, changes and deliverables.
System administrator or DevOps engineer	Prepares deployment, monitoring, backup and recovery procedures.

One person may fill several roles in a small project. The responsibilities should still be assigned, and developers should not automatically be the sole authority for approving their own work.

11. Example: Applying the Process to a PACS Probation Module

For a PACS Probation module, a practical sequence is:

13. Document existing probation processes and the problems users face.
14. Identify stakeholders and roles, including officers, station heads, sub-county, county, regional and departmental management, and IT administrators as applicable.
15. Prepare the BRD covering operational objectives, reporting needs, supervision processes and reintegration requirements.
16. Prepare the SRS with numbered functional and non-functional requirements.
17. Prepare use cases and workflows for registration, assessment, assignment, supervision, progress reporting, referrals and case closure, as applicable.
18. Approve the role-permission matrix and data visibility rules, especially for confidential records and management reporting.
19. Design the architecture, database, audit logging, reports, API and user interfaces.
20. Implement the system module by module, maintaining links between requirements, code and tests.
21. Conduct functional, security, integration, performance and UAT testing as applicable.

22. Obtain formal acceptance, deploy the system and hand over manuals, source code and maintenance documentation according to the project agreement.

For a government or public-service system, ensure applicable data-protection, records-management, security, procurement and public-sector policies are addressed in the requirements and design.

12. Key Rules for Successful Software Delivery

23. Do not start full-scale coding before requirements are understood and agreed.
24. Do not leave user roles and permissions undefined. They affect modules, database queries, reports and screens.
25. Do not approve a design based only on verbal explanations. Record the design and decisions in documents or diagrams.
26. Do not declare a feature complete merely because it works on the developer's computer. Verify it against the requirement and test cases.
27. Do not accept the final system without agreed acceptance evidence, documentation and handover deliverables.

Documentation is not paperwork for its own sake. It makes software understandable, testable, maintainable and accountable.

13. Initial Project Checklist

- ☐ Project scope and objectives approved.
- ☐ Business Requirements Document (BRD) reviewed.
- ☐ Software Requirements Specification (SRS) reviewed and approved.
- ☐ User-role and permission matrix approved.
- ☐ System design document prepared, including workflows and database design.
- ☐ Test plan and acceptance criteria agreed.

Once these core deliverables are agreed, the remaining documents can be prepared progressively in the sequence described in this guide.