

UNIT 1

System Development Fundamentals

(9 Hours) — System Analysis and Design (CAC203)

1. The Systems Development Environment

1.1 Introduction

Systems analysis and design is the process of studying an organization's current systems and procedures, identifying problems and opportunities, and then designing and building a better **information system (IS)**. An information system is an arrangement of people, data, processes, interfaces, networks and technology that collects, processes, stores and distributes information to support the operations, management and decision making of an organization.

The main people involved are the **system analyst** (bridges business and technology), **users** and **managers** (who define needs and fund the work), **programmers**, **database administrators**, **testers** and the **project manager**.

1.2 Modern Approach to Systems Analysis and Design

Early development focused on automating single tasks with little regard for integration. The modern approach treats a system as a set of interacting components serving business goals, and stresses these ideas:

- **Business orientation** — systems exist to solve business problems and create value.
- **User involvement** — users participate throughout, not only at the end.
- **Structured and iterative methods** — work is divided into phases with clear deliverables, often repeated in cycles.
- **Reuse and packaged solutions** — components, packages and services are preferred to writing everything from scratch.
- **Tool support** — CASE tools, prototyping and modelling languages accelerate and standardize the work.

1.3 Information Systems and Their Types

Type	Purpose	Typical users	Example
Transaction Processing System (TPS)	Record and process day-to-day business transactions accurately	Clerks, operational staff	Billing, payroll, ATM banking
Management Information System (MIS)	Summarize TPS data into routine reports for control	Middle managers	Monthly sales report
Decision Support System (DSS)	Analyse data and models to support semi-structured decisions	Managers, analysts	What-if budget planning
Executive Support System (ESS)	Provide high-level, strategic, often external information	Senior executives	Dashboard of KPIs and market trends
Expert System	Capture expert knowledge in rules to give advice	Specialists, non-experts	Medical diagnosis aid
Enterprise System (ERP/CRM)	Integrate functions of the whole organization	Entire organization	SAP, Odoo

2. Developing Information Systems and the SDLC

2.1 The Systems Development Life Cycle (SDLC)

The **SDLC** is a structured framework describing the phases through which a system passes from the initial idea to retirement. Each phase produces **deliverables** that feed the next. The commonly used phases are:

Phase	Key activities	Main deliverable
1. Planning	Identify the need, study feasibility, plan the project	Baseline project plan, feasibility report
2. Analysis	Determine requirements, model processes and data	Requirements statement, DFDs, ERD
3. Design	Design database, forms, reports, interfaces, architecture	Design specifications
4. Implementation	Code, test, install, document and train users	Working system, manuals, trained users
5. Maintenance	Fix, adapt and enhance the system	Change requests, new releases

2.2 The Heart of the Systems Development Process

At the centre of every approach is the repeated cycle of **analysis** → **design** → **coding** → **testing**. Whatever the model, the process is iterative: problems discovered in a later step send the team back to refine earlier work. Requirements always change, so successful methods expect and manage this loop.

2.3 The Traditional Waterfall SDLC

In the **waterfall model** the phases follow one another in strict sequence, like water falling down steps. A phase is finished and signed off before the next starts.

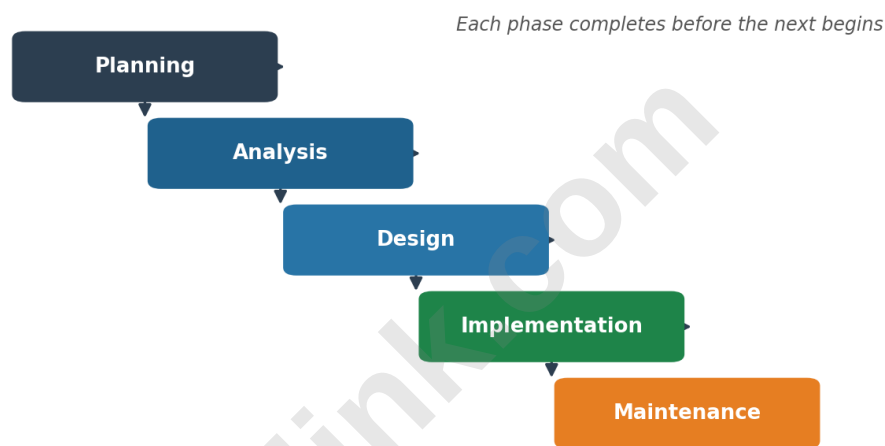


Figure 1.1 — The traditional waterfall SDLC

Advantages	Disadvantages
Simple, easy to manage and understand	Inflexible — hard to change requirements once analysis is signed off
Clear milestones and documentation	Users see the working system very late
Works well when requirements are stable and well known	Errors found late are expensive to correct

Example 1.1 — Choosing a model

A government office needs a payroll system with fixed, legally defined rules and a stable scope. Which SDLC approach suits it?

Solution

The **waterfall** model is appropriate: requirements are stable, documentation is important, and formal sign-off at each phase is required. A highly uncertain project (e.g., a new mobile start-up app) would be better served by an agile approach.

3. Approaches for Improving Development

3.1 CASE Tools

Computer-Aided Software Engineering (CASE) tools are software applications that automate or support activities of the SDLC. They improve productivity, enforce standards and keep documentation consistent.

- **Upper CASE** — support planning, analysis and design (diagramming, requirements repositories).
- **Lower CASE** — support implementation and maintenance (code generation, testing, reverse engineering).
- **Integrated CASE (I-CASE)** — cover the whole life cycle around a shared repository.

Common functions include diagramming (DFD, ERD, UML), a central **repository** of definitions, screen and report prototyping, code generation, and documentation generation. Examples: Visual Paradigm, Enterprise Architect, Lucidchart, StarUML.

3.2 Rapid Application Development (RAD)

RAD compresses development into a short time (typically 60–90 days) by using **prototyping**, **iterative development**, **CASE tools**, **reusable components** and **time-boxed** delivery with strong user participation (e.g., JAD sessions). Its phases are requirements planning, user design, construction and cutover.

Note: RAD suits systems that can be modularized; it is unsuitable for very large, high-risk or performance-critical systems.

3.3 Service-Oriented Architecture (SOA)

SOA builds applications from loosely coupled, reusable **services** that communicate over a network through standard interfaces (e.g., web services using SOAP/REST). Each service performs a business function (such as “check credit”) and can be reused by many applications. Benefits: reuse, interoperability, flexibility and easier integration of legacy systems. Costs: design complexity and network overhead. Microservices are a modern evolution of this idea.

3.4 Agile Methodologies

Agile methods develop software in short **iterations** (1–4 weeks) delivering working increments, welcoming change and relying on collaboration. The Agile Manifesto values:

We value more...	...than
Individuals and interactions	Processes and tools
Working software	Comprehensive documentation
Customer collaboration	Contract negotiation
Responding to change	Following a plan

Popular agile frameworks include **Scrum** (sprints, product backlog, daily stand-ups, roles: Product Owner, Scrum Master, Team), **Kanban** and **Extreme Programming**.

3.5 Extreme Programming (XP)

XP is an agile method emphasising technical excellence and rapid feedback. Its core practices are:

- **Pair programming** — two developers share one workstation.
- **Test-driven development** — write tests before code.
- **Continuous integration** — integrate and test many times a day.
- **Small releases** — frequent, working versions.
- **Refactoring** — continuously improve code structure.
- **Simple design and collective code ownership.**
- **On-site customer** and the **planning game** using user stories.

3.6 Object-Oriented Analysis and Design (OOAD)

OOAD models a system as a collection of interacting **objects** that combine data (attributes) and behaviour (methods). Core concepts: **class**, **object**, **encapsulation**, **inheritance**, **polymorphism**, **abstraction**. Work is documented with the **Unified Modeling Language (UML)** — use case diagrams, class diagrams, sequence diagrams and state diagrams. OOAD promotes reuse and easier maintenance and typically follows an iterative process such as the Unified Process.

Aspect	Structured approach	Object-oriented approach
Focus	Processes and data separately	Objects combining data and behaviour

Aspect	Structured approach	Object-oriented approach
Main models	DFD, ERD, structure charts	Use case, class, sequence diagrams
Reuse	Limited	High (inheritance, components)
Change	Ripples through many processes	Localized inside classes

4. The Origins of Software

4.1 Introduction and System Acquisition

Organizations no longer build every system in-house. The main sources of software are:

Source	Description	Suitable when
In-house development	Own IT staff builds custom software	Unique needs, competitive advantage
Packaged (off-the-shelf) software	Ready-made product bought from a vendor	Common needs (accounting, HR)
Outsourcing	External firm builds or runs the system	Lack of skills/capacity, cost control
Cloud / SaaS	Software rented over the internet	Low upfront cost, quick start
Open-source	Free source code that can be modified	Flexibility, community support
Enterprise solutions (ERP)	Integrated suites for the whole business	Large organizations, integration

Choosing among sources uses criteria such as cost, functionality, vendor support and viability, flexibility, documentation, response time and ease of installation. The organization normally issues a **Request for Proposal (RFP)** to vendors, evaluates the proposals against weighted criteria, and negotiates a contract.

Example 1.2 — Weighted vendor evaluation

Two vendors are scored (1–10) on Functionality (weight 0.5), Cost (0.3) and Support (0.2). Vendor A scores 8, 6, 7. Vendor B scores 7, 9, 8. Which is better?

Solution

Vendor A = $8 \times 0.5 + 6 \times 0.3 + 7 \times 0.2 = 4.0 + 1.8 + 1.4 = 7.2$

Vendor B = $7 \times 0.5 + 9 \times 0.3 + 8 \times 0.2 = 3.5 + 2.7 + 1.6 = 7.8$

Vendor B has the higher weighted score and is preferred.

4.2 Reuse

Reuse means using existing software artifacts — code, components, designs, libraries, frameworks, services — in new systems. Levels include **ad hoc** (copying), **application-level** (whole applications), **component-level** (subsystems, packages) and **object/function-level**. Reuse cuts cost and time and improves quality, but requires good documentation, a searchable repository, standards and management commitment; poorly matched components can add integration effort.

5. Managing the Information Systems Project

5.1 Introduction

A **project** is a temporary, planned effort with a defined start, end and goal. **Project management** is the application of knowledge, skills, tools and techniques to meet project requirements. The **project manager** plans, staffs, coordinates, monitors and controls the work within constraints of **scope, time, cost and quality** (the “triple constraint” plus quality).

5.2 Managing an Information Systems Project

The project management process has four phases: **initiating** (define scope, feasibility, team), **planning** (estimate, schedule, risk, resources), **executing and controlling** (direct work, track progress, manage change) and **closing** (final review, hand-over, lessons learned).

- **Project planning** — work breakdown structure (WBS), effort/time/cost estimation, resource allocation.
- **Staffing and communication** — team organization, meetings, status reports.
- **Risk management** — identify, assess, mitigate and monitor risks.
- **Change control** — evaluate and approve changes to scope.
- **Quality management** — reviews and testing.

5.3 Representing and Scheduling Project Plans

Activities are first listed in a **Work Breakdown Structure**, then sequenced and timed. Two common tools are:

Gantt chart — a horizontal bar chart showing activities against a calendar. It is easy to read but shows dependencies poorly.

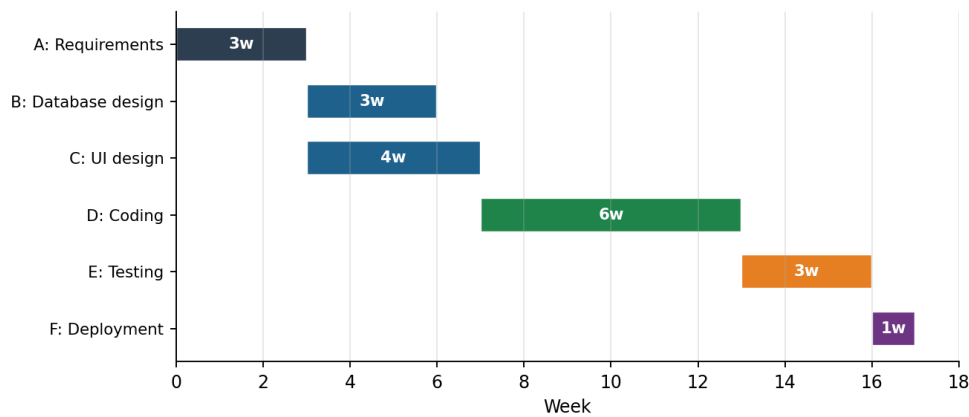


Figure 1.2 — Sample Gantt chart for a small project

Network diagram (PERT/CPM) — activities are nodes (or arrows) connected by dependencies. The **critical path** is the longest path through the network; it determines the minimum project duration, and any delay on it delays the project. Activities not on it have **slack (float)**.

Example 1.3 — Critical path

Activities: A (3 wk, no predecessor); B (3 wk, after A); C (4 wk, after A); D (6 wk, after B and C); E (3 wk, after D); F (1 wk, after E). Find the critical path and project duration.

Solution

Path A–B–D–E–F = 3 + 3 + 6 + 3 + 1 = 16 weeks

Path A–C–D–E–F = 3 + 4 + 6 + 3 + 1 = 17 weeks

The longest path is **A–C–D–E–F**, so it is the critical path and the project takes **17 weeks**. Activity B has 1 week of slack (4 – 3).

PERT estimates activity time from three values: optimistic (O), most likely (M) and pessimistic (P), using Expected time $TE = (O + 4M + P) / 6$. For O = 2, M = 5, P = 14: $TE = (2 + 20 + 14)/6 = 6$ weeks.

5.4 Using Project Management Software

Tools such as **Microsoft Project, Jira, Trello, Asana, OpenProject and Primavera** help create WBS, Gantt and network charts, assign resources, track actual against planned progress, calculate the critical path, manage costs and produce reports. Benefits include faster re-planning, better visibility and communication; limits include the need for training and the risk of over-managing.

Key Takeaways

- ✓ An information system combines people, data, processes and technology; major types are TPS, MIS, DSS, ESS and enterprise systems.
- ✓ The SDLC has planning, analysis, design, implementation and maintenance phases; waterfall runs them sequentially.
- ✓ CASE, RAD, SOA, agile, XP and OOAD are approaches to make development faster, more flexible and reusable.
- ✓ Software can be built in-house, bought, outsourced, rented (SaaS) or open source; reuse saves time and cost.
- ✓ Project management balances scope, time, cost and quality; Gantt charts and critical-path analysis schedule the work.

— *End of Unit 1* —

bcalink.com