INFORMATION SYSTEM
BCA-3RD YEAR
UNIT-3

Developing a Proposal:A **proposal** is a formal document that outlines what a software project will accomplish, how it will be done, and why it is necessary. Developing a proposal is the first critical step in the software development lifecycle as it helps stakeholders understand the scope, costs, benefits, and risks involved.
The proposal serves as a roadmap that guides decision-making and resource allocation. It must be clear, convincing, and realistic.

Feasibility Study

Before committing resources, it is essential to assess whether the project is **feasible**. Feasibility study answers the question: *Can we build this system, and is it worth building?*

Types of Feasibility:

a) Technical Feasibility

- Examines if current technology, hardware, software, and technical expertise can support the project.
- Asks:
    - Do we have the right tools?
    - Is the technology mature or experimental?
    - Can existing systems integrate with the new system?

Example:

If a company wants a cloud-based application but lacks internet infrastructure, technical feasibility may be low.

b) Economic Feasibility (Cost-Benefit Analysis)

- Analyzes costs versus expected financial benefits.
- Includes initial investment, operational costs, and long-term savings or revenue.
- Tools like ROI (Return on Investment) or NPV (Net Present Value) may be used.

Example:

If developing a software costs $100,000 but saves $150,000 annually in labor, it may be economically feasible.

c) Operational Feasibility

- Assesses whether the system will function effectively within the existing organizational environment.
- Considers human resources, organizational culture, and user acceptance.
- Checks if staff are willing and able to use the new system.

d) Legal Feasibility

- Ensures compliance with laws, regulations, and licensing.
- Example: Data protection laws may affect how user data is handled.

e) Schedule Feasibility

- Evaluates whether the project timeline is realistic given resource constraints.
- Delays can cause increased costs or lost business opportunities.

How to Conduct a Feasibility Study

1. **Understand the Project Scope:** Review requirements and objectives.
2. **Identify Key Constraints:** Budget, technology, legal, timeline.
3. **Analyze Each Feasibility Area:** Technical, economic, operational, legal, schedule.
4. **Gather Data:** Interviews, research, expert opinions.
5. **Prepare Feasibility Report:** Summarize findings, recommendations.

Sample Feasibility Report Template

Section	Description
Project Overview	Brief description of the project
Technical Feasibility	Assessment of technology resources and needs
Economic Feasibility	Cost estimates and financial benefits
Operational Feasibility	Organizational fit and resource availability
Legal Feasibility	Compliance with laws and regulations
Schedule Feasibility	Timeline and deadline considerations
Conclusion and Recommendation	Summary of findings and go/no-go decision

Cost Estimation

Cost estimation is predicting the total expenses involved in the software project. It helps in budgeting, planning, and evaluating project viability.

Types of Costs

Cost Type	Description
Development Cost	Salaries of developers, designers, testers
Hardware/Software Cost	Purchase of equipment, licenses
Training Cost	Training users and staff
Maintenance Cost	Bug fixes, updates after release
Overhead	Office space, utilities, administration

Cost Estimation Techniques

- **Expert Judgment:** Experienced managers estimate based on knowledge.
- **Analogous Estimation:** Use costs from similar past projects.
- **Parametric Estimation:** Uses mathematical models (e.g., cost per function point or line of code).
- **Bottom-up Estimation:** Estimate each task individually and sum.

Example: Cost Estimation for a Banking App

- Developer salaries: $200,000
- Hardware/software licenses: $50,000
- Training: $10,000

- Maintenance (first year): $30,000
- Total estimated cost = $290,000

Cost Estimation Techniques

- **Expert Judgment:** Experienced managers estimate based on knowledge.
- **Analogous Estimation:** Use costs from similar past projects.
- **Parametric Estimation:** Uses mathematical models (e.g., cost per function point or line of code).
- **Bottom-up Estimation:** Estimate each task individually and sum.

Example: Cost Estimation for a Banking App

- Developer salaries: $200,000
- Hardware/software licenses: $50,000
- Training: $10,000
- Maintenance (first year): $30,000
- Total estimated cost = $290,000

Overview of System Design

System design transforms requirements into a blueprint for building the system. It defines architecture, data, processes, interfaces, and controls.

6. Design of Input and Control

Input design ensures users enter data correctly and easily.

- Use **forms** with logical layouts.
- Add **input validation** to prevent errors (e.g., numeric only, mandatory fields).
- Use **dropdown lists** or **radio buttons** to limit choices.
- Provide clear **instructions and feedback** on input errors.

Control design enforces rules and security:

- Access controls for sensitive data.
- Confirmation dialogs before critical actions.
- Validation checks.

7. Design of Output and Control

Outputs communicate information to users or other systems.

- **Reports:** Daily sales report, error logs.
- **On-screen display:** Dashboards, alerts.
- **Data export:** CSV files, APIs.

Design outputs to be:

- Clear and organized.
- Timely.
- Secure (restrict sensitive data access).

8. File Design / Database Design

Focuses on organizing data storage for efficiency and integrity.

Data Modeling (ER Diagram Example)

- **Entities:** Customer, Order, Product.

- **Attributes:** Customer (Name, ID), Order (Date, Amount).
- **Relationships:** Customer places Order.

Normalization: Normalization is a systematic process in database design that organizes data to:

- **Minimize** redundancy (duplicate data),
- Avoid insertion, update, and deletion anomalies (errors),
- Ensure data integrity and logical consistency.

It involves decomposing large tables into smaller, well-structured tables without losing data or relationships.

Why Normalize?

Imagine a customer order database that stores customer name, address, and orders in the same table. If a customer changes their address, you'd have to update multiple records — this leads to errors.

Normalization solves this by organizing data into logical tables, reducing duplication, and simplifying maintenance.

Normal Forms (NFs)

Normalization is defined in terms of normal forms, which are rules or conditions a table must satisfy to reduce redundancy and anomalies.

The most commonly used normal forms are:

- **1NF:** Atomic values, no repeating groups.
- **2NF:** No partial dependency on part of primary key.
- **3NF:** No transitive dependencies.

Indexing

Speeds up data retrieval.

9. Process Design

Describes the logic and flow of data processing.

- Use **Flowcharts** to visualize steps.
- Use **DFDs** to show data movement.
- Include **error handling**.

10. User Interface Design

- Prioritize usability and accessibility.
- Use consistent design elements.
- Provide helpful feedback.
- Design simple navigation.

11. Prototyping

- Build early models.
- Obtain user feedback.
- Refine requirements and design.

12. Software Constructors

- Tools like IDEs, code generators, and reusable libraries.

3. Documentation

- Includes user manuals and technical documents.
- Essential for maintenance and training.

What is Process Design?

Process Design in software engineering refers to the detailed planning and specification of the workflows, algorithms, and logical operations that transform inputs into desired outputs within a system.

It defines **how data moves through the system**, how it is processed, and how decisions are made. The goal is to create an efficient, clear, and maintainable process flow that meets business requirements.

Key objectives of process design:

- Specify the sequence of operations.
- Define decision points and data transformations.
- Support business rules and logic.
- Handle exceptions and errors gracefully.

Tools for Process Design

Two popular visual tools to represent processes are:

1. Flowcharts

- **Definition:**
  A flowchart is a graphical diagram that shows the step-by-step flow of control or data through processes using standardized symbols.
- **Common Symbols:**
  - **Oval:** Start or End
  - **Rectangle:** Process or Operation
  - **Diamond:** Decision point (Yes/No)
  - **Arrow:** Flow direction
  - **Parallelogram:** Input/Output
- **Example Flowchart:**

Imagine designing a process for user login:

Start → Enter Username & Password → Check Credentials → (Decision)
  ├─ If valid → Grant Access → End
  └─ If invalid → Show Error → Retry → End

In flowchart form, this shows each step and decision clearly.

2. Data Flow Diagrams (DFDs)

- **Definition:**
  DFDs visually represent how data moves through a system, focusing on the flow between processes, data stores, and external entities.
- **Key Components:**

- **Process:** Transforms data (circle or rounded rectangle).
- **Data Store:** Where data is stored (open-ended rectangle).
- **External Entity:** Source or destination outside the system (square).
- **Data Flow:** Arrows showing data movement.

- **Example DFD for Order Processing:**

Customer (External Entity) → (Place Order) Process → Order Database (Data Store)

↓

(Generate Invoice) Process → Customer

DFDs help understand system boundaries and data dependencies.

Business Rules and Logic Design

What are Business Rules?

Business rules are the specific conditions, constraints, and policies that govern how business processes operate. They define the logic for decision making within the system.

Examples:

- A customer's credit limit must not be exceeded.
- Orders over $500 require manager approval.
- Users must change passwords every 90 days.

Logic Design

Logic design involves translating business rules into executable system logic, which can be implemented using:

- Decision tables.
- Conditional statements (if-else).
- Algorithms and procedures.

Example: Logic for Order Approval

Condition	Action
Order amount ≤ $500	Approve automatically
Order amount > $500 and manager available	Send to manager approval
Order amount > $500 and no manager available	Hold order

This decision table clearly captures the business rule logic.

What is Exception Handling?

Exception handling is designing how the system responds to unexpected events or errors during process execution. This ensures the system can recover gracefully without crashing or producing incorrect results.

Why is Exception Handling Important?

- Prevents data corruption.
- Enhances system reliability.
- Improves user experience by providing meaningful error messages.

- Ensures security by managing unauthorized access or invalid input.

Common Exception Handling Strategies

- **Error Detection:** Identify when something goes wrong (e.g., invalid input, system failure).
- **Error Reporting:** Inform the user or system admin clearly.
- **Recovery:** Attempt to correct the problem or rollback transactions.
- **Logging:** Record errors for future debugging.

Example: Exception Handling in User Input

- If a user enters a non-numeric value in an age field:
    - Detect invalid input.
    - Show message: “Please enter a valid number.”
    - Prompt user to re-enter.
    - Log the incident.

Summary

Aspect	Description
Process Design	Planning workflows and logic to transform inputs to outputs.
Flowcharts	Visual diagrams showing process steps and decisions.
Data Flow Diagrams	Visualize movement of data between processes and stores.
Business Rules	Policies and constraints that control decision making.
Logic Design	Implementation of business rules into system logic.
Exception Handling	Managing errors and unexpected events gracefully.

Purpose of Prototyping

- To create an **early model** or **mock-up** of the software system or some components.
- Helps users and developers **visualize** requirements.
- Allows early **feedback** to refine requirements and design.
- Reduces misunderstanding between stakeholders.
- Helps explore design alternatives.
- Minimizes costly changes late in development.

Types of Prototyping

1. **Throwaway (Rapid) Prototyping**
    - Build quickly to clarify requirements.
    - Prototype is discarded after feedback.
    - Final system is developed from scratch.
2. **Evolutionary Prototyping**

    - Prototype is continuously refined.
    - Eventually evolves into the final product.
    - Useful when requirements are unclear or evolving.
3. **Incremental Prototyping**
    - Different parts of the system are prototyped separately.
    - Later integrated into the full system.
4. **Extreme Prototyping** (mainly for web apps)
    - Consists of three phases: static prototype, simulation of services, and fully functional prototype.

Steps to Create a Prototype

1. **Identify Basic Requirements:**
   Gather high-level user needs.
2. **Develop Initial Prototype:**
   Build a quick version focusing on key features.
3. **User Evaluation:**
   Present prototype to users for feedback.
4. **Refine Prototype:**
   Modify design based on feedback.
5. **Repeat:**
   Iterate the cycle until requirements are clear.
6. **Develop Final System:**
   Use the refined prototype as a basis or discard it (depending on type).

Advantages of Prototyping

- Improves user involvement and satisfaction.
- Helps detect misunderstandings early.
- Facilitates requirement clarification.
- Reduces risk of project failure.
- Encourages exploration of alternative designs.

Disadvantages of Prototyping

- Can increase development time if too many iterations.
- Users may expect prototype as the final system, causing confusion.
- May result in poor design if prototype is not properly evolved.
- Resource-intensive if thrown away repeatedly.

Tools and Technologies for Prototyping

- **UI/UX Design Tools:**
  Figma, Adobe XD, Sketch, Balsamiq.
- **Rapid Application Development (RAD) Tools:**
  Microsoft PowerApps, OutSystems.
- **Programming Frameworks:**
  React, Angular, Flutter for interactive prototypes.

- **Mockup Tools:**
  MockFlow, Axure RP.

Software Constructors

Definition

Software Constructors are tools, libraries, and utilities that assist developers in creating software more efficiently by automating or simplifying parts of the development process.

Types of Software Constructors

- **Integrated Development Environments (IDEs):**
  Tools like Visual Studio, Eclipse that provide code editing, debugging, and project management.
- **Code Generators:**
  Automatically generate boilerplate code from models or specifications.
- **Reusable Libraries and Frameworks:**
  Pre-built modules that provide common functionalities (e.g., authentication libraries).
- **Component-Based Tools:**
  Allow building software by assembling pre-made components.

Role in Software Development

- Increase productivity by reducing manual coding.
- Enforce consistency and coding standards.
- Help avoid common errors by using tested components.
- Support rapid development, prototyping, and maintenance.

Examples

Type	Examples
IDE	Visual Studio, IntelliJ IDEA
Code Generator	Yeoman, Swagger Codegen
Libraries/Frameworks	React.js, Spring Framework
Component Tools	Qt Designer, Delphi Components

Benefits of Using Constructors

- Faster development cycles.
- Improved software quality.
- Easier maintenance due to modular code.
- Encourages reuse of tested software components.
- Helps new developers onboard quickly.

Documentation

Types of Documentation

1. **User Documentation:**
   Manuals, help guides, FAQs for end-users.
2. **System Documentation:**
   Technical specifications, architecture diagrams, API docs for developers and maintainers.
3. **Process Documentation:**
   Development process, testing procedures, project plans.
4. **Training Documentation:**
   Tutorials, workshops, onboarding materials.

Importance of Documentation

- Facilitates software maintenance and upgrades.
- Provides knowledge transfer between team members.
- Helps users understand and effectively use the software.
- Serves as a reference during audits or compliance checks.
- Reduces dependency on original developers.

Best Practices for Documentation

- Keep it **clear, concise, and up-to-date**.
- Use **standardized templates** for consistency.
- Include **diagrams and examples** for better understanding.
- Use **version control** to track changes.
- Make documentation **easily accessible** to all stakeholders.
- Review and update regularly.

Sample Documentation Structure

Section	Content Description
Introduction	Overview and purpose of the software
System Architecture	High-level design and component descriptions
Installation Guide	Steps to install and configure the software
User Guide	How end-users can operate the system
API Documentation	Details on programmatic interfaces
Maintenance Procedures	Backup, recovery, troubleshooting steps
Glossary	Definitions of technical terms
Revision History	Log of changes to the documentation