UNIT I - Concept of System (Exam-Oriented Notes)

Introduction

In Software Engineering, a system is the foundation for understanding how software is designed. A system is a collection of interrelated components that work together to achieve a common objective. Before developing software, engineers study the existing system, its inputs, outputs, processes and interactions.

Definition

A system is an organized collection of interconnected components or subsystems that interact with one another to achieve a common goal.

Basic System Diagram

INPUT
 |
 v
PROCESSING
 |
 v
OUTPUT
 |
 v
FEEDBACK

Components of a System

Input

Data or resources supplied to the system, e.g., ATM card, student details.

Processing

Operations performed to convert input into meaningful information.

Output

Final result produced after processing, such as a receipt or marksheet.

Feedback

Information used to improve or control future performance.

Characteristics of a System

- Organization: Components are arranged systematically.
- Interaction: Components communicate with each other.
- Interdependence: One subsystem depends on another.
- Integration: Modules combine to form one complete system.
- Goal-oriented: Every system is built to achieve a specific objective.

ATM Example

Input: ATM card, PIN, amount.
Processing: Verify user, check balance, deduct amount.
Output: Cash, receipt, updated balance.
Feedback: SMS and transaction history.

Advantages

- Improves efficiency
- Reduces manual work
- Provides accurate results
- Saves time
- Easy management

Exam-Oriented Questions

- Define a system.
- Explain the components of a system with a neat diagram.
- Describe the characteristics of a system.

Programmes vs Software Products & Emergence of Software Engineering

1. Programmes vs Software Products

A programme is a set of instructions written to solve a specific problem. A software product is a complete software solution delivered to users. It includes programs, documentation, testing, installation, maintenance and user support.

Programme	Software Product
Single task	Complete solution
Small size	Large and complex
Single developer	Team development
Minimal testing	Extensive testing
Limited documentation	Complete documentation
No regular maintenance	Regular updates and maintenance

2. Emergence of Software Engineering

In the early days, software was small and written by individual programmers. As business and scientific applications became larger, projects suffered from delays, budget overruns, poor quality and maintenance problems. This period became known as the Software Crisis. Software Engineering emerged to apply engineering principles, defined processes, documentation, testing and project management to software development.

Flow:
Small Programs → Large Systems → Software Crisis → Software Engineering

Reasons for Emergence

- Increasing software size
- Growing complexity
- High development cost
- Late delivery
- Poor quality
- Maintenance difficulties
- Need for standard development processes

3. Early Computer Programming

Early computers were programmed in Machine Language and Assembly Language. Machine language used binary digits and was difficult to understand. Assembly language introduced mnemonics but remained hardware-dependent.

Limitations

- Difficult coding
- Time-consuming
- Error-prone
- Hard to debug
- Not portable

4. High-Level Language Programming

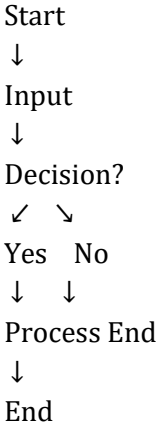
High-level languages such as FORTRAN, COBOL, C, C++, Java and Python allow programmers to write readable, portable and maintainable programs. Compilers and interpreters translate these languages into machine code.

Advantages

- Easy to learn
- Portable
- Faster development
- Better maintenance
- Supports large applications

5. Control Flow Based Design

Control flow based design focuses on the order of execution of statements using sequence, selection and iteration. It promotes structured programming.



Advantages

- Simple
- Easy debugging
- Logical flow

6. Data Structure Oriented Design

This approach begins by designing data structures before designing algorithms. Proper organization of data leads to simpler and more maintainable software.

- Focus on data
- Improves maintainability
- Suitable for information systems

7. Object Oriented Design

Object-Oriented Design (OOD) models software using classes and objects. It improves software reuse, modularity and maintainability.

- Encapsulation
- Inheritance
- Polymorphism
- Abstraction

```
Class Car
 |-- speed
 |-- start()
     |
  Object: myCar
```

Exam Questions

- Differentiate between Programme and Software Product.
- Explain the Software Crisis and emergence of Software Engineering.
- Discuss Early Computer Programming and High-Level Languages.
- Explain Control Flow Based Design.
- Explain Data Structure Oriented Design.
- Describe Object-Oriented Design with advantages.
-

UNIT II:

Evolutionary Model

Evolutionary Model develops software in multiple versions. Instead of waiting until the end, a usable version is delivered early and then improved repeatedly using customer feedback. This approach is suitable when requirements are expected to change or are not completely known at the beginning.

Definition

Evolutionary Model develops software in multiple versions. Instead of waiting until the end, a usable version is delivered early and then improved repeatedly using customer feedback. This approach is suitable when requirements are expected to change or are not completely known at the beginning.

Working Process

1. Requirements collection
2. Develop initial version
3. Customer evaluation
4. Add new features
5. Repeat until complete

Flow Diagram:
Start
↓
Requirements collection
↓
Develop initial version
↓
Customer evaluation
↓
Add new features
↓
Repeat until complete
↓
End

Advantages

- Accommodates changing requirements
- Early delivery
- Continuous customer feedback
- Lower risk than one-time delivery

Disadvantages

- Requires customer involvement
- Management complexity
- May increase project duration

Applications

- ERP systems
- E-commerce websites
- Business applications

Exam Tips

- Draw the flow diagram of Evolutionary Model.
- Remember two advantages and two disadvantages of Evolutionary Model.
- Write one real-life application.

Spiral Model

Spiral Model, proposed by Barry Boehm, combines iterative development with systematic risk analysis. Each loop of the spiral represents one development cycle and begins with planning and ends with customer evaluation.

Definition

Spiral Model, proposed by Barry Boehm, combines iterative development with systematic risk analysis. Each loop of the spiral represents one development cycle and begins with planning and ends with customer evaluation.

Working Process

6. Planning
7. Risk analysis
8. Engineering (design, coding, testing)
9. Customer evaluation
10. Next iteration

Flow Diagram:
Start
↓
Planning
↓
Risk analysis
↓
Engineering (design, coding, testing)
↓
Customer evaluation
↓
Next iteration
↓
End

Advantages

- Excellent risk management
- Suitable for large projects
- Frequent customer feedback
- Supports changes

Disadvantages

- High cost
- Needs risk experts

- Not suitable for small projects

Applications

- Defense
- Banking
- Air traffic control
- Medical systems

Exam Tips

- Draw the flow diagram of Spiral Model.
- Remember two advantages and two disadvantages of Spiral Model.
- Write one real-life application.

Agile Model

Agile is an iterative and incremental software development methodology that emphasizes customer collaboration, working software, continuous improvement and rapid delivery through short iterations called sprints.

Definition

Agile is an iterative and incremental software development methodology that emphasizes customer collaboration, working software, continuous improvement and rapid delivery through short iterations called sprints.

Working Process

11. Product backlog
12. Sprint planning
13. Sprint
14. Testing
15. Review
16. Retrospective
17. Next sprint

Flow Diagram:
Start
↓
Product backlog
↓
Sprint planning
↓
Sprint
↓
Testing
↓
Review
↓
Retrospective
↓
Next sprint
↓
End

Advantages

- Fast delivery
- High customer satisfaction

- Flexible
- Continuous testing

Disadvantages

- Needs experienced teams
- Less documentation
- Frequent requirement changes

Applications

- Mobile apps
- Web applications
- Startups
- Cloud software

Exam Tips

- Draw the flow diagram of Agile Model.
- Remember two advantages and two disadvantages of Agile Model.
- Write one real-life application.

UNIT III - SOFTWARE PLANNING

1. Introduction to Software Planning

Software planning is the process of deciding how a software project will be executed, monitored and completed. It includes estimating project size, cost, schedule, resources, risks and quality objectives. Good planning reduces uncertainty and increases the probability of delivering software on time and within budget.

Planning Flow:
Requirements
↓
Size Estimation
↓
Effort Estimation
↓
Scheduling
↓
Resource Allocation
↓
Risk Management
↓
Project Execution

2. Responsibilities of a Software Project Manager

- Define project scope and objectives.
- Prepare project plan and schedule.
- Estimate cost, effort and project size.
- Allocate human and technical resources.
- Identify and manage project risks.
- Monitor project progress using metrics.
- Maintain communication with stakeholders.
- Ensure software quality through reviews and testing.
- Manage budget and project documentation.
- Deliver the project on time.

3. Metrics for Project Size Estimation

Project size estimation predicts the amount of software to be developed. Accurate estimation helps in planning cost, schedule and staffing. Two common metrics are Lines of Code (LOC) and Function Point (FP).

4. LOC (Lines of Code)

LOC measures software size by counting the number of executable source code lines. It is language-dependent and generally calculated after design or coding.

Formula

Productivity = LOC / Effort (person-months)

Advantages

- Simple to understand.
- Useful for productivity analysis.
- Easy for completed projects.

Disadvantages

- Language dependent.
- Cannot be estimated accurately early.
- Ignores software functionality.

Example

If a project contains 20,000 source lines of code and requires 10 person-months, productivity = 20,000 / 10 = 2,000 LOC per person-month.

5. Function Point (FP) Metric

Function Point measures software size based on the functionality delivered to the user instead of source code. It can be estimated during requirements analysis and is independent of programming language.

- External Inputs (EI)
- External Outputs (EO)
- External Inquiries (EQ)
- Internal Logical Files (ILF)
- External Interface Files (EIF)

Basic Steps

- Identify the five function types.
- Assign complexity weights.
- Calculate Unadjusted Function Points (UFP).

- Apply adjustment factor (if required).
- Obtain final Function Point count.

Advantages

- Language independent.
- Can be estimated early.
- Suitable for business applications.
- Useful for comparing projects.

Disadvantages

- Requires trained estimators.
- Calculation can be subjective.

Comparison: LOC vs Function Point

Feature	LOC	Function Point
Basis	Source code lines	User functionality
Language	Dependent	Independent
When estimated	Late	Early
Best for	Coding productivity	Business/project estimation

Exam-Oriented Questions

- Define software planning.
- Explain the responsibilities of a Software Project Manager.
- Explain LOC metric with advantages and disadvantages.
- Explain Function Point metric.

Compare LOC and Function Point metrics. Project Estimation Techniques, COCOMO Model & SRS

1. Project Estimation Techniques

Project estimation is the process of predicting the effort, time, cost, staffing and resources required to complete a software project. Accurate estimation helps managers prepare realistic schedules and budgets.

Objectives

- Estimate project cost
- Estimate development time

- Estimate team size
- Reduce project risk
- Improve planning

Common Estimation Techniques

- Expert Judgment
- Analogous Estimation
- Top-Down Estimation
- Bottom-Up Estimation
- Lines of Code (LOC)
- Function Point (FP)
- COCOMO Model

2. COCOMO (Constructive Cost Model)

COCOMO, developed by Barry Boehm, estimates software development effort, development time and cost based on project size measured in KLOC (thousands of lines of code).

COCOMO Project Types

- Organic: Small, experienced teams and simple projects.
- Semi-Detached: Medium-sized projects with mixed experience.
- Embedded: Complex projects with strict hardware/software constraints.

Basic COCOMO Formula

Effort (Person-Months) = a × (KLOC)^b
Development Time (Months) = c × (Effort)^d
The constants a, b, c and d depend on the project type.

Mode	a	b	c	d
Organic	2.4	1.05	2.5	0.38
Semi-Detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Worked Example

Example: For an Organic project of 32 KLOC:
Effort = 2.4 × (32)^1.05 ≈ 91 person-months (approx.).
Development time = 2.5 × (91)^0.38 ≈ 14 months (approx.).

Advantages

- Simple and widely used
- Supports planning

- Provides effort and schedule estimates

Limitations

- Depends on KLOC estimate
- Less suitable for Agile
- Accuracy depends on input estimates

3. Software Requirement Specification (SRS)

A Software Requirement Specification (SRS) is a formal document that describes all functional and non-functional requirements of a software system. It acts as an agreement between the customer and the development team.

Purpose of SRS

- Defines system requirements
- Acts as a communication document
- Provides a basis for design, testing and maintenance
- Reduces misunderstandings

Typical Contents of an SRS

- Introduction
- Overall description
- Functional requirements
- Non-functional requirements
- External interfaces
- Constraints
- Assumptions
- Appendices

Characteristics of a Good SRS

- Correct
- Complete
- Consistent
- Unambiguous
- Verifiable
- Modifiable
- Traceable
- Feasible
- Ranked by importance
- Easy to understand

Importance of SRS

- Improves communication
- Reduces development errors
- Helps estimation
- Supports testing
- Simplifies maintenance

Exam-Oriented Questions

- Explain project estimation techniques.
- Explain the COCOMO model with formula.
- Differentiate Organic, Semi-Detached and Embedded modes.
- Define SRS and explain its characteristics.
- Why is SRS important in software engineering?

UNIT IV - Software Design and Implementation Notes

1. Software Design

Software design is the process of transforming software requirements into an architecture and detailed design that can be implemented in code. A good design improves quality, maintainability and reusability.

Characteristics of Good Software Design

- Correct
- Simple
- Modular
- Efficient
- Reliable
- Maintainable
- Reusable
- Scalable
- Flexible
- Secure

Cohesion

Cohesion measures how closely related the responsibilities of a module are. High cohesion is desirable.

- Coincidental
- Logical
- Temporal
- Procedural
- Communicational
- Sequential
- Functional (best)

Coupling

Coupling measures dependency between modules. Low coupling is desirable.

- Content (worst)
- Common
- External
- Control
- Stamp
- Data (best)

Cohesion vs Coupling

Aspect	Cohesion	Coupling
Meaning	Within module	Between modules
Goal	High	Low
Effect	Better maintenance	Lower dependency

Function-Oriented Design

Breaks software into functions. Uses DFD, data dictionary, decision tables and decision trees.

Data Flow Diagram

DFD shows movement of data.

Entity --> Process --> Data Store --> Output

Data Dictionary

Repository describing data elements, structures and meanings.

Decision Table

Tabular representation of conditions and actions.

Decision Tree

Tree showing decisions and outcomes.

Object-Oriented Design

OOD organizes software into classes and objects using encapsulation, inheritance, polymorphism and abstraction.

Structured Coding Techniques

- Top-down design
- Modular programming
- Meaningful names
- Avoid goto
- Single responsibility

Coding Styles

- Indentation
- Comments
- Naming conventions
- Consistent formatting
- Error handling

Documentation

- Internal documentation (comments)
- External documentation (SRS, user manual, API guide)

Exam Questions

- Explain characteristics of good software design.
- Explain cohesion and coupling.
- Explain DFD and Data Dictionary.
- Differentiate function-oriented and object-oriented design.

- Explain coding styles and documentation.

UNIT V - SOFTWARE TESTING

1. Concept of Software Testing

Software testing is the systematic process of evaluating software to identify defects and verify that it satisfies specified requirements. Testing improves quality, reliability, security and user satisfaction. The primary objective is to detect defects before software is released.

Objectives

- Detect defects
- Verify requirements
- Validate user needs
- Improve quality
- Reduce maintenance cost

Basic Flow:
Requirements -> Design -> Coding -> Testing -> Bug Fixing -> Release

2. Verification vs Validation

Verification checks whether the software is built correctly according to specifications. Validation checks whether the correct software has been built to satisfy user needs.

Feature	Verification	Validation
Question	Are we building the product right?	Are we building the right product?
Focus	Specifications	User requirements
Performed by	Developers/QA	Testing team & users
Techniques	Reviews, inspections	Execution, acceptance testing

3. Unit Testing

Unit testing verifies individual functions, methods or modules independently. It is usually performed by developers before integration.

- Tests smallest software unit

- Uses test cases
- Helps early bug detection
- Supports regression testing

Example: Test a login() function separately before combining with the complete application.

4. Black Box Testing

Black box testing evaluates software functionality without examining internal source code. Test cases are based on inputs, outputs and requirements.

- No knowledge of code required
- Focuses on functionality
- Common techniques: Equivalence Partitioning, Boundary Value Analysis

Advantages

- User-oriented
- Independent of programming language
- Finds missing functionality

Limitations

- Cannot test internal logic
- May miss hidden paths

5. White Box Testing

White box testing examines the internal structure, logic and source code. Test cases are designed to cover statements, branches and paths.

- Requires programming knowledge
- Statement coverage
- Branch coverage
- Path coverage

Advantages

- Thorough logic testing
- Finds coding errors

Limitations

- Time-consuming
- Needs skilled testers

6. Integration Testing

Integration testing verifies interactions between integrated modules after unit testing. It detects interface defects.

- Top-down integration
- Bottom-up integration
- Big-bang integration
- Sandwich/Hybrid integration

7. System Testing

System testing evaluates the complete integrated system against functional and non-functional requirements before delivery.

- Functional testing
- Performance testing
- Security testing
- Usability testing
- Recovery testing

8. Introduction to Configuration Management

Software Configuration Management (SCM) controls changes to software artifacts throughout the software life cycle. It maintains version history, prevents conflicts and supports team collaboration.

- Version control
- Change management
- Build management
- Release management
- Configuration audits

Comparison: Black Box vs White Box

Aspect	Black Box	White Box
Knowledge of code	Not required	Required
Focus	Functionality	Internal logic
Performed by	QA/Testers	Developers/Testers
Coverage	Requirements	Statements/Branches

Important University Questions

- Define software testing and explain its objectives.
- Differentiate verification and validation.
- Explain unit testing.
- Compare black box and white box testing.
- Explain integration testing approaches.
- Explain system testing.
- What is Software Configuration Management?

Comparison of Software Life Cycle Models

Feature	Waterfall	Iterative	Evolutionary	Spiral	Agile	Prototype
Requirement changes	Poor	Good	Excellent	Excellent	Excellent	Excellent
Risk handling	Low	Medium	Medium	Very High	Medium	Low
Customer involvement	Low	Medium	High	High	Very High	Very High
Cost	Low	Medium	Medium	High	Medium	Medium
Suitable for	Stable	Medium	Changing req.	Large critical	Fast changing	Unclear req.

Important University Questions

- Explain Evolutionary Model with diagram.
- Explain Spiral Model with neat diagram.
- Explain Agile Model and its principles.
- Compare different Software Life Cycle Models.