Unit I: Fundamentals of Software Engineering

1. Software Scope

- **Defines project boundaries and objectives.**
- **Clearly articulates functionalities.**
- **Establishes operational constraints.**
- **Documents key design assumptions.**
- **Provides a basis for estimating project costs, timelines, and resources.**
- **Helps manage stakeholder expectations and reduce scope creep.**

NEXT POINT

2. The Software Process

- **Structured activities for developing software.**
- **Includes:**
  - **Requirement Analysis: Gathering and documenting needs.**
  - **Design: Creating architecture and specifications.**
  - **Coding/Implementation: Writing code.**
  - **Testing: Verifying functionality.**
  - **Deployment: Releasing to users.**
  - **Maintenance: Ongoing updates and fixes.**

NEXT POINT

3. Software Myths

- **Misconceptions that can lead to project failures.**
- **Includes:**
  - **"Adding more programmers speeds up development" (Brooks' Law).**

- **"Once the software is developed, the job is done."**
- **"Good software doesn't need documentation."**
- **"Requirements will not change."**

NEXT POINT

4. Software Engineering Discipline

- **Systematic and quantifiable approach.**
- **Draws from traditional engineering and computer science.**
- **Emphasizes methodical problem-solving.**
- **Focuses on reliable, maintainable, and efficient software.**
- **Incorporates best practices, standards, and metrics.**
- **Balances technical excellence with business constraints.**

NEXT POINT

5. Generic Process Model and Related Process Models

- **Common framework for projects.**
- **Phases: Requirement Analysis → Design → Implementation → Testing → Deployment → Maintenance.**
- **Models:**
    - **Waterfall Model: Linear, sequential, for stable requirements.**
    - **Incremental Model: Develops in chunks with added features.**
    - **Spiral Model: Risk-driven, combines waterfall and prototyping.**
    - **V-Model: Emphasizes verification and validation at each stage.**
    - **Agile Model: Iterative, customer-focused, responds to change.**

NEXT POINT

6. Software Development Challenges

- **Changing Requirements**
- **Budget and Time Constraints**
- **Security Vulnerabilities**
- **Maintaining Software Quality**
- **Scalability and Performance Issues**
- **Technical Debt**
- **Integration with Legacy Systems**

NEXT POINT

7. Software Methodologies - *Expanded Explanations*

- **Frameworks for organizing the development process.**
    - **Waterfall:**
        - **Sequential approach moving from requirements to maintenance.**
        - **Emphasizes comprehensive documentation and formal review processes.**
        - **Works well for projects with stable, well-understood requirements.**
        - **Lacks flexibility when requirements change.**

      Next MODEL

    - **Agile:**
        - **Embraces iterative development, adaptive planning, and frequent delivery.**
        - **Prioritizes customer collaboration, team interaction, and responding to change.**
        - **Includes frameworks like Scrum, Kanban, and XP (Extreme Programming).**

Next MODEL

- **Spiral:**
  - **Risk-driven methodology combining elements of waterfall and prototyping.**
  - **Each spiral cycle addresses progressively more complete versions of the software.**
  - **Manages risks effectively.**
  - **Useful for large, complex systems with significant uncertainty.**

Next MODEL

- **V-Model:**
  - **Associates a testing phase with each development stage.**
  - **Unit tests verify detailed design, integration tests verify architectural design, system tests verify system requirements.**
  - **Creates a V-shaped workflow emphasizing verification and validation.**

Next MODEL

- **RAD (Rapid Application Development):**
  - **Focuses on fast prototyping and iterative delivery to gather feedback quickly.**
  - **Relies heavily on tools and techniques that accelerate development, like code generators, visual programming, and reusable components.**

NEXT POINT

8. Current Challenges in Software Development and Status

- **Current Challenges:**
    - **Cybersecurity Threats**
    - **Managing Large-scale Distributed Systems**
    - **Integrating AI and Automation**
    - **Handling Big Data**
    - **Compliance with Privacy Laws (GDPR, CCPA)**
- **Current Status:**
    - **Increasing Adoption of Agile and DevOps**
    - **AI and ML Integration**
    - **Rise of Cloud Computing**
    - **Containerization and Microservices**
    - **Low-Code/No-Code Development**

NEXT POINT

9. Introduction to Agile Software Engineering

- **Shift in development philosophy.**
- **Emphasizes:**
    - **Iterative and Incremental Development**
    - **Adaptive Planning**
    - **Customer Collaboration**
    - **Working Software Over Documentation**
    - **Individuals and Interactions**
- **Frameworks:**
    - **Scrum: Sprints, daily stand-ups, defined roles.**
    - **Kanban: Workflow visualization, limits work in progress.**

- **Extreme Programming (XP): Test-driven development, pair programming.**

NEXT POINT

10. Examples of Activities in Each Phase of SDLC

- **Requirement Gathering: Stakeholder interviews, user personas, use case diagrams, surveys, observation, analysis.**
- **Design: System architecture, database schemas, UI/UX wireframes, component interfaces, coding standards, technical specifications.**
- **Implementation: Coding, unit tests, component integration, environment setup, code reviews.**
- **Testing: Unit testing, integration testing, system testing, performance testing, user acceptance testing, security testing.**
- **Deployment: Production environment preparation, deployment scripts, data migration, monitoring configuration, pre-production testing, rollback procedures.**
- **Maintenance: Bug fixes, minor enhancements, performance monitoring, security patches, performance optimization, technical debt management.**

NEXT UNIT

Unit II: Requirement Analysis

1. Requirements Capturing

- **Identifying and documenting software needs from users, stakeholders, and the business.**
- **Serves as the basis for development activities.**
- **Involves diverse stakeholders.**
- **Uses techniques like interviews, workshops, and observation.**
- **Captures both explicit and implicit requirements.**

- **Identifies constraints and assumptions.**
- **Establishes acceptance criteria.**

NEXT POINT

2. Requirements Engineering

- **Systematic approach to gathering, analyzing, documenting, and managing requirements.**
- **Key activities:**
    - **Elicitation: Extracting requirements using interviews, surveys, workshops, and observation.**
    - **Specification: Documenting requirements in a clear and verifiable manner.**
    - **Validation: Confirming requirements are correct, complete, and aligned with business goals.**
    - **Negotiation: Resolving conflicts between stakeholder needs.**
    - **Prioritizing Requirements: Determining essential versus nice-to-have requirements.**

NEXT POINT

3. Customer Problem Statement

- **Articulates the issue the software aims to solve.**
- **Defines the issue without implying a specific solution.**
- **Identifies who experiences the problem and how it affects them.**
- **Quantifies the problem to demonstrate its scale.**
- **Includes context clarifying the problem's relevance.**

NEXT POINT

4. User Stories

- **Simple descriptions of a feature from the user's perspective.**
- **Format: "As a [user type], I want [goal] so that [benefit]."**
- **Example: "As a customer, I want to be able to track my order so that I know when it will arrive."**

NEXT POINT

5. MoSCoW Method

- **Prioritization technique for requirements.**
- **Categories:**
    - **Must have: Critical for project success.**
    - **Should have: Important but not vital.**
    - **Could have: Desirable but not necessary.**
    - **Won't have: Not planned for this release.**

NEXT POINT

6. Kano Analysis

- **Analyzes customer satisfaction based on requirement fulfillment.**
- **Categories:**
    - **Basic Expectations: Expected features; absence leads to dissatisfaction.**
    - **Performance Needs: Features that increase satisfaction as they improve.**
    - **Excitement Generators: Unexpected features that delight customers.**

NEXT POINT

7. Real-Life Application Case Study (MoSCoW, Kano)

- **Applying MoSCoW and Kano to real-world projects.**
- **Example: E-commerce platform prioritizing features based on customer needs.**

NEXT POINT

8. Software Requirement Specification (SRS) Introduction

- **Comprehensive document detailing all software requirements.**
- **Sections:**
    - **Introduction: Purpose, scope, definitions.**
    - **Overall Description: Product perspective, functions, characteristics.**
    - **Specific Requirements: Functional, non-functional, interface requirements.**
    - **Support Information: Appendices, index.**

NEXT POINT

9. Introduction to: Miro/Jira for Storymap Requirements Analysis

- **Using Miro/Jira for visualizing and managing requirements.**
- **Creating story maps to organize user stories and tasks.**

NEXT POINT

10. Requirements Analysis: Basics

- **Analyzing documented requirements for clarity, completeness, and consistency.**
- **Techniques include:**

- **Decomposition: Breaking down complex requirements.**
- **Classification: Grouping requirements by category.**
- **Prioritization: Determining importance.**

NEXT POINT

11. Scenario-based Modeling

- **Creating scenarios to understand how users interact with the system.**
- **Use cases: Describe interactions between users and the system.**

NEXT POINT

12. UML Models

- **Unified Modeling Language for visualizing software systems.**
  - **Use Case Diagram: Describes interactions between users and the system.**
  - **Class Diagram: Shows the structure of the system, classes, and relationships.**
  - **Activity Diagram: Illustrates workflows and activities.**
  - **State Diagrams: Shows the states of an object and transitions between them.**
  - **Sequence Diagrams: Illustrates interactions between objects over time.**

NEXT POINT

13. Real-Life Application Case Study (UML)

- **Applying UML models to real-world projects.**
- **Example: Modeling an online banking system using UML diagrams.**

14. Use of StarUML

- **Using StarUML tool for creating UML diagrams.**

END