ECE 811 – SOFTWARE ENGINEERING

WEEK 1: INTRODUCTION TO SOFTWARE ENGINEERING - STUDY GUIDE/REVISION

I. FUNDAMENTAL CONCEPTS

1. What is Software Engineering?

- Software engineering is the systematic application of engineering principles to software design, development, and maintenance.
- Goals: Reliable, efficient, scalable, and maintainable software.

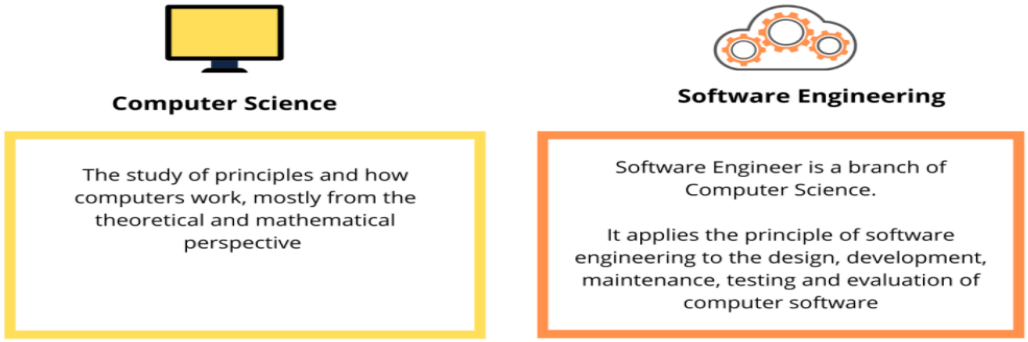

Figure 1. Computer science and Software engineering

2. **What is Software Crisis** refers to the challenges faced in developing efficient and useful computer programs due to increasing complexity and demands. Factors like poor project management, inadequate training, and low productivity contribute to this crisis. Addressing these issues through systematic approaches like software engineering, with a focus on budget control, quality, timeliness, and skilled workforce, can mitigate the impact of the crisis.
3. **Industrial strength software system** is software designed and built to be exceptionally reliable, robust, and scalable, capable of handling demanding workloads and critical operations within an organization.

4. Key Principles

Key principles of software engineering include modularity, abstraction, encapsulation, reusability, and maintainability. These principles guide the development of software that is reliable, maintainable, and scalable. They also emphasize efficient project management, collaboration, and adherence to best practices throughout the software lifecycle. These features are expounded below.

a) **Abstraction**: Hiding of unnecessary implementation details and exposing only the essential functionality. This simplifies usage and allows for changes without affecting other parts of the system.

b) **Modularity**: Breaking down a complex system into smaller, independent, and manageable modules. This allows for easier development, testing, and maintenance..

c) **Encapsulation**: Bundling data and methods that operate on that data within a single unit (like a class). This protects data from unintended external access and modification.

d) Reusability: Designing components that can be used in multiple contexts or projects, saving time and effort.

e) Maintainability: Designing software that is easy to understand, modify, and debug. This includes using clean code, proper documentation, and following established design patterns.

f) **Testing:** Verifying that the software meets requirements and is free of defects. This includes unit testing, integration testing, and system testing.

5. **Software Quality Attributes**
    - Correctness
    - Robustness
    - Performance
    - Maintainability
    - Usability
    - Scalability

II. SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC)

Model	Key Idea	Pros	Cons
Waterfall	Linear phases (Requirements → Maintenance)	Simple, structured	Inflexible; late feedback
Iterative	Cycles of prototyping + refinement	Early feedback; risk reduction	Scope creep risk
Agile	Incremental delivery in sprints	Adaptable; customer-centric	Less documentation
DevOps	Integrates development + operations	Faster deployment; automation	Cultural shift required

Agile Manifesto Values: Check the website

III. CORE PROCESSES

1. **Requirements Engineering**
    - **Elicitation**: Interviews, surveys, user stories.
    - **Specification**: SRS document (functional/non-functional reqs).
    - **Validation**: Prototyping, feasibility analysis.
2. **Software Design**

- **Architectural Patterns**:
  - MVC (Model-View-Controller)
  - Client-Server, Microservices, Layered
- **Design Patterns**: Singleton, Factory, Observer (Gang of Four).
- **UML Diagrams**: Use case, class, sequence diagrams.

3. **Implementation & Version Control**
   - **Coding Standards**: Readability, naming conventions.
   - **Git Essentials**: commit, branch, merge, pull request.
   - **Refactoring**: Improving code without changing behavior.
4. **Testing & QA**
   - **Levels**: Unit → Integration → System → Acceptance.
   - **Techniques**:
     - White-box (code structure) ✗ Black-box (functionality)
     - Automated testing (JUnit, Selenium).
   - **CI/CD**: Jenkins, GitHub Actions.
5. **Maintenance**
   - **Types**: Corrective (bug fixes), Adaptive (OS updates), Perfective (features), Preventive (optimization).
   - **Legacy Systems**: Challenges in maintaining outdated software.

IV. PROJECT MANAGEMENT

1. **Team Roles**
   - Product Owner
   - Scrum Master
   - Developers
   - QA Engineers.
2. **Estimation Techniques**
   - COCOMO model
   - Planning Poker ✗ Function Points.
3. **Risk Management**
   - Identify → Analyze → Plan → Monitor.
4. **Tools**
   - Jira (issue tracking) ✗ Trello (kanban) ✗ Slack (communication).

V. SOFTWARE QUALITY & ETHICS

1. **Quality Assurance**
    - Code reviews ✗ Static analysis (SonarQube) ✗ ISO/IEC 9126 standards.
2. **Metrics**
    - Cyclomatic complexity
    - Code coverage
    - Bug density.
3. **Professional Ethics**
    - ACM/IEEE Code of Ethics: Privacy, intellectual property, social responsibility.

VI. MODERN PRACTICES

1. **Cloud-Native Development**
    - Containers (Docker) ✗ Orchestration (Kubernetes).
2. **AI in SE**
    - Automated code generation ✗ Bug prediction.
3. **Secure Development**
    - OWASP Top 10
    - DevSecOps integration.