SNS COLLEGE OF TECHNOLOGY

Coimbatore-35.

An Autonomous Institution

COURSE NAME : 19CST201 AGILE SOFTWARE ENGINEERING

II YEAR/ III SEMESTER

UNIT – I INTRODUCTION TO SOFTWARE ENGINEERING

UNIT I INTRODUCTION TO SOFTWARE ENGINEERING

The Nature of Software -Software Engineering - Software engineering Practice – Process Models: Generic – Prescriptive – Specialized - United Process - Personal and Team Process Models - Process Technology-Understanding Requirements-Design concepts & model-Software quality concepts & Review metrics.

Design Concepts

- Abstraction
- Architecture
- Patterns
- Separation of Concerns
- Modularity
- Information Hiding

Design Concepts

- Functional Independence
- Refinement
- Aspects
- Refactoring
- Object-Oriented Design Concepts
- Design Classes

Design Concepts

➢ Abstraction

- Solution to any problem can have many levels of abstraction can be used.
- At the **highest level of abstraction**, a solution is stated in broad terms using the language of the problem environment.
- At **lower levels of abstraction**, a more detailed description of the solution is provided. Problem-oriented terminology is coupled with implementation-oriented terminology in an effort to state a solution.
- Finally, at the **lowest level of abstraction**, the solution is stated in a manner that can be directly implemented.
- A **procedural abstraction** refers to a sequence of instructions that have a specific and limited function.
- A **data abstraction** is a named collection of data that describes a data object.

Design Concepts

➢ Architecture

- Software architecture - the overall structure of the software and the ways in which that structure provides conceptual integrity for a system.

- **Structural properties -** This aspect of the architectural design representation defines the components of a system (e.g., modules, objects, filters) and the manner in which those components are packaged and interact with one another.

- **Extra-functional properties.** The architectural design description should address how the design architecture achieves requirements for performance, capacity, reliability, security, adaptability, and other system characteristics.

- **Families of related systems.** The architectural design should draw upon repeatable patterns that are commonly encountered in the design of families of similar systems. In essence, the design should have the ability to reuse architectural building blocks.

Design Concepts

➢ Architecture

- With these properties, the architectural design can be represented using one or more of a number of different models :
- ***Structural models*** **-** represent architecture as an organized collection of program components.
- ***Framework models*** **-** increase the level of design abstraction by attempting to identify repeatable architectural design frameworks that are encountered in similar types of applications.
- ***Dynamic models*** **-** address the behavioral aspects of the program architecture, indicating how the structure or system configuration may change as a function of external events.
- ***Process models*** **-** focus on the design of the business or technical process that the system must accommodate. Finally,
- ***Functional models*** – can be used to represent the functional hierarchy of a system.

Design Concepts

➢ Patterns

- The intent of each design pattern is to provide a description that enables a designer to determine

- (1) whether the pattern is applicable to the current work,
- (2) whether the pattern can be reused
- (3) whether the pattern can serve as a guide for developing a similar, but functionally or structurally different pattern.

Design Concepts

➢ Separation of Concerns

- ***Separation of concerns*** is a design concept that suggests that any complex problem can be more easily handled if it is subdivided into pieces that can each be solved and/or optimized independently.
- **A *concern*** is a feature or behavior that is specified as part of the requirements model for the software.

➢ Modularity

- Modularity is the most common manifestation of separation of concerns. Software is divided into separately named and addressable components, sometimes called *modules*, that are integrated to satisfy problem requirements.

➢ Information Hiding

Design Concepts

Functional Independence :

- Functional independence is achieved by developing functions that perform only one kind of task and do not excessively interact with other modules.
- Independence is important because it makes implementation more accessible and faster.
- The independent modules are easier to maintain, test, and reduce error propagation and can be reused in other programs as well.
- Independence is assessed using two qualitative criteria: cohesion and coupling.

- ***Cohesion*** is an indication of the relative functional strength of a module.

- ***Coupling*** is an indication of the relative interdependence among modules.

Design Concepts

➢ Refinement :

- Stepwise refinement is a top-down design strategy originally proposed by Niklaus Wirth.
- A program is developed by successively refining levels of procedural detail.
- A hierarchy is developed by decomposing a macroscopic statement of function (a procedural abstraction) in a stepwise fashion until programming language statements are reached.
- Refinement helps you to reveal low-level details as design progresses.

Design Concepts

➢ Aspects

- A feature linked to many other parts of the program, but which is not related to the program's primary function. An aspect crosscuts the program's core concerns, therefore violating its separation of concerns that tries to encapsulate unrelated functions.

➢ Refactoring

- *Refactoring* is a reorganization technique that simplifies the design (or code) of a component without changing its function or behavior.

- Fowler defines refactoring - “Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code [design] yet improves its internal structure.”

Design Concepts

➢ Object-Oriented Design Concepts

- Classes and objects, inheritance, messages, and polymorphism.

➢ Design Classes

- *Design classes* that refine the analysis classes by providing design detail that will enable the classes to be implemented, and implement a software infrastructure

Design Concepts

Types of Design Classes :

- ***User interface classes*** define all abstractions that are necessary for human computer interaction (HCI).

- ***Business domain classes*** identify the attributes and services (methods) that are required to implement some element of the business domain.

- ***Process classes*** implement lower-level business abstractions required to fully manage the business domain classes.

- ***Persistent classes*** represent data stores (e.g., a database) that will persist beyond the execution of the software.

- ***System classes*** implement software management and control functions that enable the system to operate and communicate within its computing environment and with the outside world.

Design Model

1. Data Design Elements
2. Architectural Design Elements
3. Interface Design Elements
4. Component-Level Design Elements
5. Deployment-Level Design Elements

Design Model

1. Data Design Elements

- Data design (sometimes referred to as *data architecting*) creates a model of data and/or information that is represented at a high level of abstraction (the customer/user's view of data).

- This data model is then refined into progressively more implementation-specific representations that can be processed by the computer-based system.

Design Model

2. **Architectural Design Elements** :

- The *architectural design* for software is the equivalent to the floor plan of a house.
- The floor plan depicts the overall layout of the rooms; their size, shape, and relationship to one another; and the doors and windows that allow movement into and out of the rooms
- The architectural model is derived from three sources:

(1) information about the application domain for the software to be built;
(2) specific requirements model elements such as data flow diagrams or analysis classes, their relationships and collaborations for the problem at hand;
(3) the availability of architectural styles

Design Model

3. **Interface Design Elements :**

- The interface design for software is analogous to a set of detailed drawings (and specifications) for the doors, windows, and external utilities of a house.

- These drawings depict the size and shape of doors and windows, the manner in which they operate.

- There are three important elements of interface design:

(1) the user interface (UI);
(2) external interfaces to other systems, devices, networks, or other producers or consumers of information
(3) internal interfaces between various design components.

Design Model

4. Component-Level Design Elements:

- The component-level design for software is the equivalent to a set of detailed drawings (and specifications) for each room in a house. These drawings depict wiring and plumbing within each room, the location of electrical receptacles and wall switches, cabinets, and closets.

- The component-level design defines data structures for all local data objects and algorithmic detail for all processing

Design Model

5. Deployment-Level Design Elements:

- Deployment-level design elements indicate how software functionality and subsystems will be allocated within the physical computing environment that will support the software.

Software Quality Concepts

1. What Is Quality?

2. Software Quality - *An effective software process applied in a manner that creates a useful product that provides measurable value for those who produce it and those who use it.*

- Garvin's Quality Dimensions
- McCall's Quality Factors
- ISO 9126 Quality Factors
- Targeted Quality Factors
- The Transition to a Quantitative View

Software Quality Concepts

➢ Garvin's Quality Dimensions :

- David Garvin suggests that quality should be considered by taking a multidimensional viewpoint.

- **Performance quality.** Does the software deliver all content, functions, and features that are specified as part of the requirements model in a way that provides value to the end user?

- **Feature quality.** Does the software provide features that surprise and delight first-time end users?

- **Reliability.** Does the software deliver all features and capability without failure? Is it available when it is needed? Does it deliver functionality that is error-free?

Software Quality Concepts

➢ Garvin's Quality Dimensions :

- **Conformance.** Does the software conform to local and external software standards that are relevant to the application? Does it conform to de facto design and coding conventions? For example, does the user interface conform to accepted design rules for menu selection or data input?

- **Durability.** Can the software be maintained (changed) or corrected (debugged) without the inadvertent generation of unintended side effects? Will changes cause the error rate or reliability to degrade with time?

- **Serviceability.** Can the software be maintained (changed) or corrected (debugged) in an acceptably short time period? Can support staff acquire all information they need to make changes or correct defects?

Software Quality Concepts

➢ McCall's Quality Factors:

- ***Correctness*** - The extent to which a program satisfies its specification and fulfills the customer's mission objectives.
- ***Reliability*** - The extent to which a program can be expected to perform its intended function with required precision. [It should be noted that other, more complete definitions of reliability have been proposed.
- ***Efficiency*** - The amount of computing resources and code required by a program to perform its function.
- ***Integrity*** - Extent to which access to software or data by unauthorized persons can be controlled.
- ***Usability*** - Effort required to learn, operate, prepare input for, and interpret output of a program.

Software Quality Concepts

➢ McCall's Quality Factors:

- ***Maintainability***. Effort required to locate and fix an error in a program
- ***Flexibility***. Effort required to modify an operational program.
- ***Testability***. Effort required to test a program to ensure that it performs its intended function.
- ***Portability***. Effort required to transfer the program from one hardware and/or software system environment to another.
- ***Reusability***. Extent to which a program [or parts of a program] can be reused in other applications—related to the packaging and scope of the functions that the program performs.
- ***Interoperability***. Effort required to couple one system to another.

Software Quality Concepts

- **ISO 9126 Quality Factors:**

- **Functionality**
- **Reliability**
- **Usability**
- **Efficiency**
- **Maintainability**
- **Portability**

Software Quality Concepts

➢ Targeted Quality Factors :

- **Efficiency**
- **Robustness -** The degree to which the software handles bad input data or inappropriate user interaction
- **Richness -** The degree to which the interface provides a rich feature set.

➢ The Transition to a Quantitative View

Software Quality Concepts

3. The Software Quality Dilemma
    - “Good Enough” Software
    - The Cost of Quality
    - Risks
    - Negligence and Liability
    - Quality and Security
    - The Impact of Management Actions

Software Quality Concepts

4. Achieving Software Quality
    - Software Engineering Methods
    - Project Management Techniques
    - Quality Control
    - Quality Assurance

Software Quality Concepts

Project Management Techniques :

(1) A project manager uses estimation to verify that delivery dates are achievable,

(2) Schedule dependencies are understood and the team resists the temptation to use short cuts,

(3) Risk planning is conducted

Review Metrics

Review Metrics

- Analyzing Metrics
- Cost Effectiveness of Reviews

Review Metrics

Review Metrics :

- ***Preparation effort, Ep***—the effort (in person-hours) required to review a work product prior to the actual review meeting
- ***Assessment effort, Ea***—the effort (in person-hours) that is expended during the actual review
- ***Rework effort, Er*** —the effort (in person-hours) that is dedicated to the correction of those errors uncovered during the review

Review Metrics

Review Metrics :

- ***Work product size, WPS*** —a measure of the size of the work product that has been reviewed
- ***Minor errors found, Errminor*** —the number of errors found that can be categorized as minor
- ***Major errors found, Errmajor*** —the number of errors found that can be categorized as major

Review Metrics

1. Analyzing Metrics :

- The total review effort and the total number of errors discovered are defined as:

$$E_{review} = E_p + E_a + E_r$$

$$\text{Err}_{tot} = \text{Err}_{minor} + \text{Err}_{major}$$

Error density represents the errors found per unit of work product reviewed.

$$\text{Error density} = \frac{\text{Err}_{tot}}{\text{WPS}}$$


Thank You!