**SEPM**

[Software Engineering | SDLC V-Model - GeeksforGeeks](https://www.geeksforgeeks.org/software-engineering-sdlc-v-model/)

[Incremental Model (Software Engineering) - javatpoint](https://www.javatpoint.com/software-engineering-incremental-model)

[Software Engineering | Evolutionary Model - GeeksforGeeks](https://www.geeksforgeeks.org/software-engineering-evolutionary-model/#:~:text=Evolutionary%20model%20is%20a%20combination,envisioning%20need%20to%20be%20done).

[Agile Model (Software Engineering) - javatpoint](https://www.javatpoint.com/software-engineering-agile-model)

[Software Engineering | Requirements Engineering Process - GeeksforGeeks](https://www.geeksforgeeks.org/software-engineering-requirements-engineering-process/)

[Types of Feasibility Study in Software Project Development - GeeksforGeeks](https://www.geeksforgeeks.org/types-of-feasibility-study-in-software-project-development/)

[Software Engineering | Prototyping Model - GeeksforGeeks](https://www.geeksforgeeks.org/software-engineering-prototyping-model/)

[Overview Software Documentation - GeeksforGeeks](https://www.geeksforgeeks.org/overview-software-documentation/)

[Software Requirement Specification (SRS) Format - GeeksforGeeks](https://www.geeksforgeeks.org/software-requirement-specification-srs-format/)

[The Management Spectrum | 4 P's in Software Project Planning - GeeksforGeeks](https://www.geeksforgeeks.org/4-ps-in-software-project-planning/)

[Software Engineering | Software Metrics - javatpoint](https://www.javatpoint.com/software-engineering-software-metrics)

[Lines of Code (LOC) in Software Engineering - GeeksforGeeks](https://www.geeksforgeeks.org/lines-of-code-loc-in-software-engineering/)

**Module 3**

Design Process & quality,

Good software design should exhibit:

Firmness: A program should not have any bugs that inhibit its function.

Commodity: A program should be suitable for the purposes for which it was intended.

Delight: The experience of using the program should be pleasurable one.

Quality Attributes

Functionality

Usability

Reliability

Performance

Supportability

Maintainability

Extensibility

Adaptability

Serviceability

Testability

Compatibility

Configurability

Design Concepts

* Abstraction —data, procedure, control ~ fundamental way to reduce complexity
* Architecture —the overall structure of the software
* Patterns — “conveys the essence” of a proven design solution
* Separation of concerns —any complex problem can be more easily handled if it is subdivided into pieces
* Modularity —compartmentalization of data and function
* Hiding—controlled interfaces
* Functional independence —outgrowth of Modularity
* Refinement —top-down elaboration of detail for all abstractions
* Refactoring —a reorganization technique that simplifies the design
* Design Classes —provide design detail that will enable analysis classes to be implemented

The design Model elements

Data elements

Data model --> data structures

Data model --> database architecture

Architectural elements

Information about application domain

Analysis classes, their relationships, collaborations and behaviors are transformed into design realizations

Patterns and “styles”

Interface elements

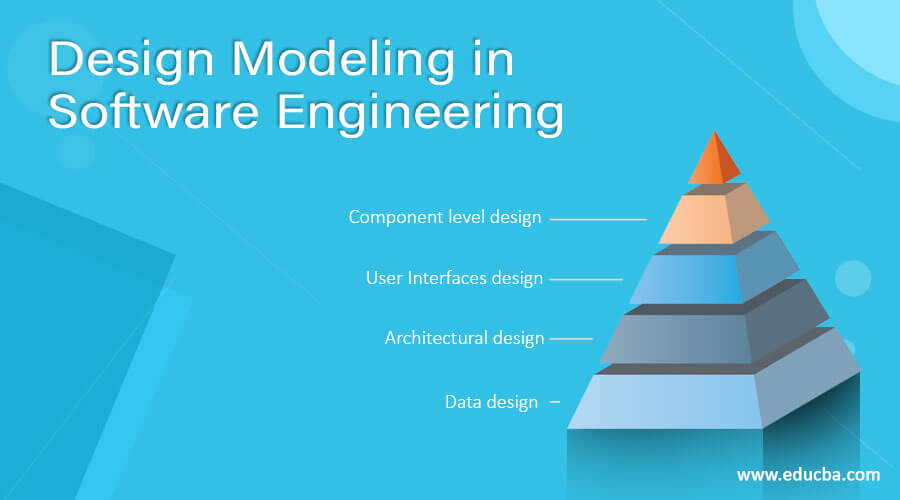
The user interface (UI)

External interfaces to other systems, devices, networks or other producers or consumers of information

Internal interfaces between various design components.

Component elements

Deployment elements\



Architectural Design:

software components, the externally visible properties of those components, and the relationships among them.

Components, connectors, constraints, semantic models

* Data-centered architecture
* Data flow architecture
* Call and return architecture
* Object-oriented architecture
* Layered architecture

Modeling Component level Design:

1. Identify design classes in problem domain
2. Identify infrastructure design classes
3. Elaborate design classes
4. Describe persistent data sources
5. Elaborate behavioral representations
6. Elaborate deployment diagrams
7. Refactor design and consider alternatives

Designing class based components,

conducting component-level design

Component-level design defines the **data structures, algorithms, interface characteristics, and communication mechanisms** allocated to each component

User Interface Design:

The golden rules

* Place the user in control
* Reduce the user’s memory load
* Make the interface consistent

Interface Design steps & Analysis

User Analysis Can be done through :

1. User interviews
2. Sales Team Input
3. Marketing Team Input
4. Support Team Input

Design Issues

* Response time
* Help facilities
* Error handling
* Menu and command labeling
* Application accessibility
* Internationalization

**Module 4 :**

**Software Risk,**

**Configuration Management**

The art of coordinating software development to minimize confusion is called configuration management.

identifying, organizing, and controlling modifications

Maximize productivity by minimizing confusion

The items that comprise all information produced as part of the software process are collectively called a software configuration.

Software Configuration Items (SCI)

* + Computer programs

both source level and executable forms

* + Documents (that describe the computer programs)

targeted at both technical practitioners and users

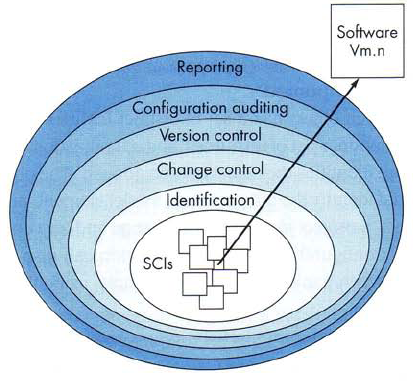
* + Data

contained within the program or external to it

SCM repositories Functions

* Versioning
* Dependency Tracking and Change Management
* Requirements Tracing
  + Forward Tracking : Ability to track all the design components and deliverables that result from a specific requirement
  + Backward Tracking : Ability to identify which requirement generated any given deliverable
* Configuration management
  + keeps track of a series of configurations representing specific project milestones or production releases.
* Audit trails
  + establishes additional information about when, why, and by whom changes are made.

SCM Process



**SQA**

Risk Identification, Risk Assessment, Risk Projection, RMMM Software Configuration management



Risk Management

* Reactive
* Proactive

Goals

Requirements quality

The correctness, completeness, and consistency of the requirements model will have a strong influence on the quality of all work products that follow.

Design quality

The software team should assess every element of the design model to ensure that it exhibits high quality and that the design itself conforms to requirements.

Code quality

Source code and related work products (e.g., other descriptive information) must conform to local coding standards and exhibit characteristics that will facilitate maintainability.

Quality control effectiveness

A software team should apply limited resources in a way that has the highest likelihood of achieving a high-quality result.

Formal Tecnical Review (FTR)

the audience of customers, management, and technical staff.

Objectives

1. To uncover errors in function, logic, or implementation for any representation of the software
2. To verify that the software under review meets its requirements
3. To ensure that the software has been represented according to predefined standards
4. To achieve software that is developed in a uniform manner
5. To make projects more manageable

Steps

Review, Reporting, and Record keeping