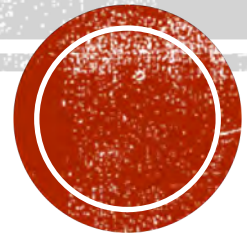
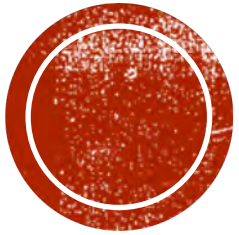
SOFTWARE DESIGN

Software Engineering

Dr. Raj Singh





DESIGN CONCEPTS

SOFTWARE DESIGN

Encompasses the set of principles, concepts, and practices that lead to the development of a high quality system or product

Design principles establish and overriding philosophy that guides the designer as the work is performed

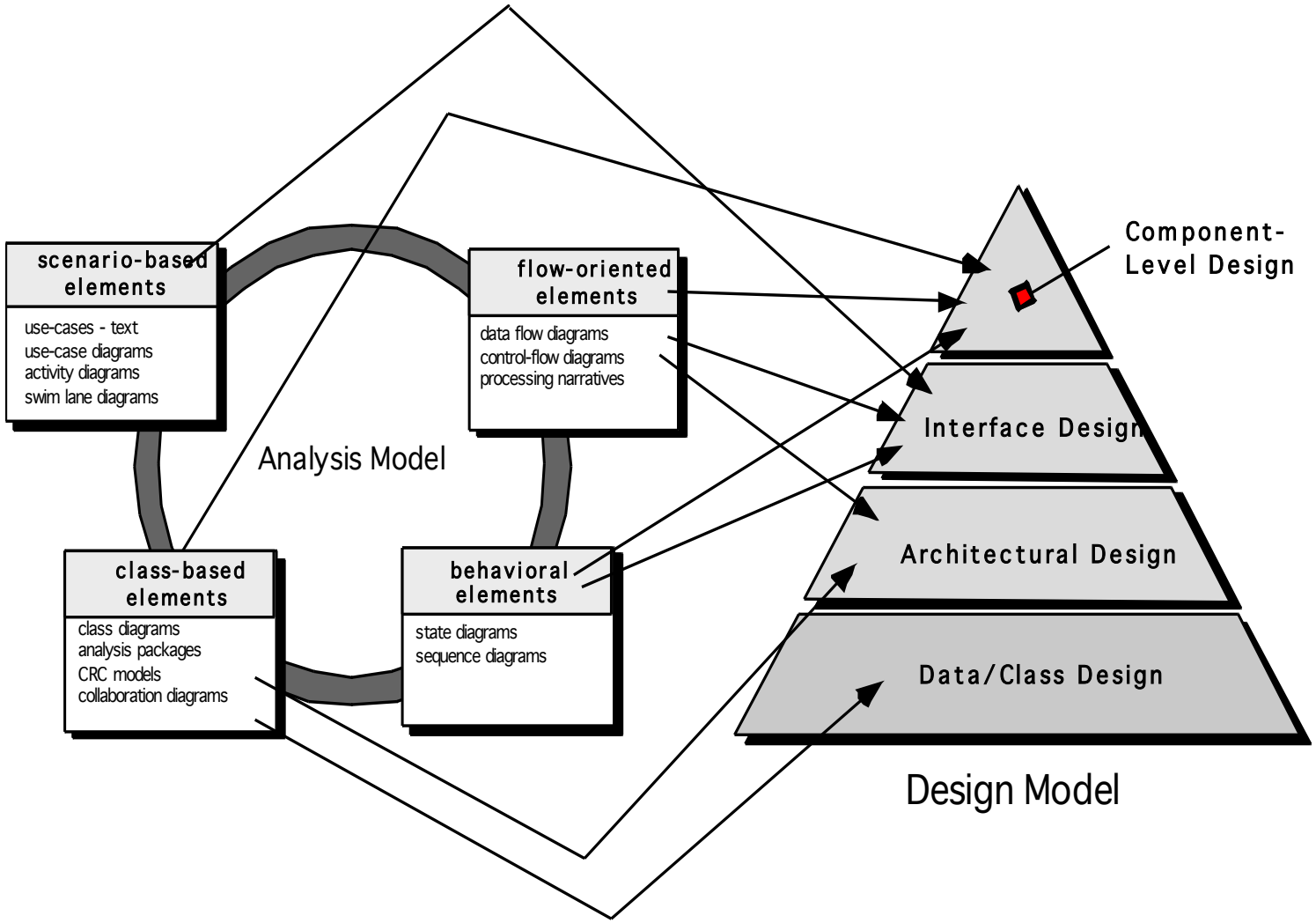
Design concepts must be understood before the mechanics of design practice are applied

Software design practices change continuously as new methods, better analysis, and broader understanding evolve.

SOFTWARE ENGINEERING DESIGN

User Interface design	defines how software elements, hardware elements, and end-users communicate
Architectural design	defines relationships among the major software structural elements
Component-level design	transforms structural elements into procedural descriptions of software components
Data/Class design	transforms analysis classes into implementation classes and data structures

ANALYSIS MODEL → DESIGN MODEL



DESIGN AND QUALITY

A good design leads to good quality product

The design must implement all of the explicit and implicit requirements contained in the requirements analysis model

The design should be

Easy to understand and follow.

Provide a complete picture of the software to be implemented.

Flexible, scalable, modular, uniform, maintainable.

FUNDAMENTAL DESIGN CONCEPTS

Abstraction
- data, procedure, control

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

Information hiding
- controlled interfaces

Functional independence
- single-minded function and low coupling

Refinement
- elaboration of detail for all abstractions

Aspects
- a mechanism for understanding how global requirements affect design

Refactoring
- a reorganization technique that simplifies the design

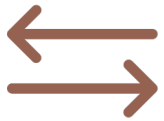
OO DESIGN CONCEPTS

Design classes

Entity, boundary, and controller classes

Inheritance

all responsibilities of a superclass is immediately inherited by all subclasses



Messages

stimulate some behavior to occur in the receiving object

Polymorphism

a characteristic that greatly reduces the effort required to extend the design

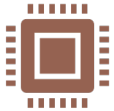
DESIGN MODEL ELEMENTS

Data

Data model → data structures, database architecture

Architectural

Application domain, classes, relationships, collaborations, behaviors, patterns



Interface

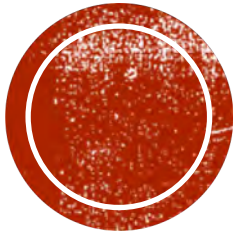
User interface (UI), components, other systems, devices, networks

Component

APIs, business and service logic

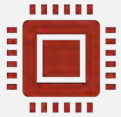
Deployment

Continuous integration and deployement



ARCHITECTURAL DESIGN

WHY ARCHITECTURE?



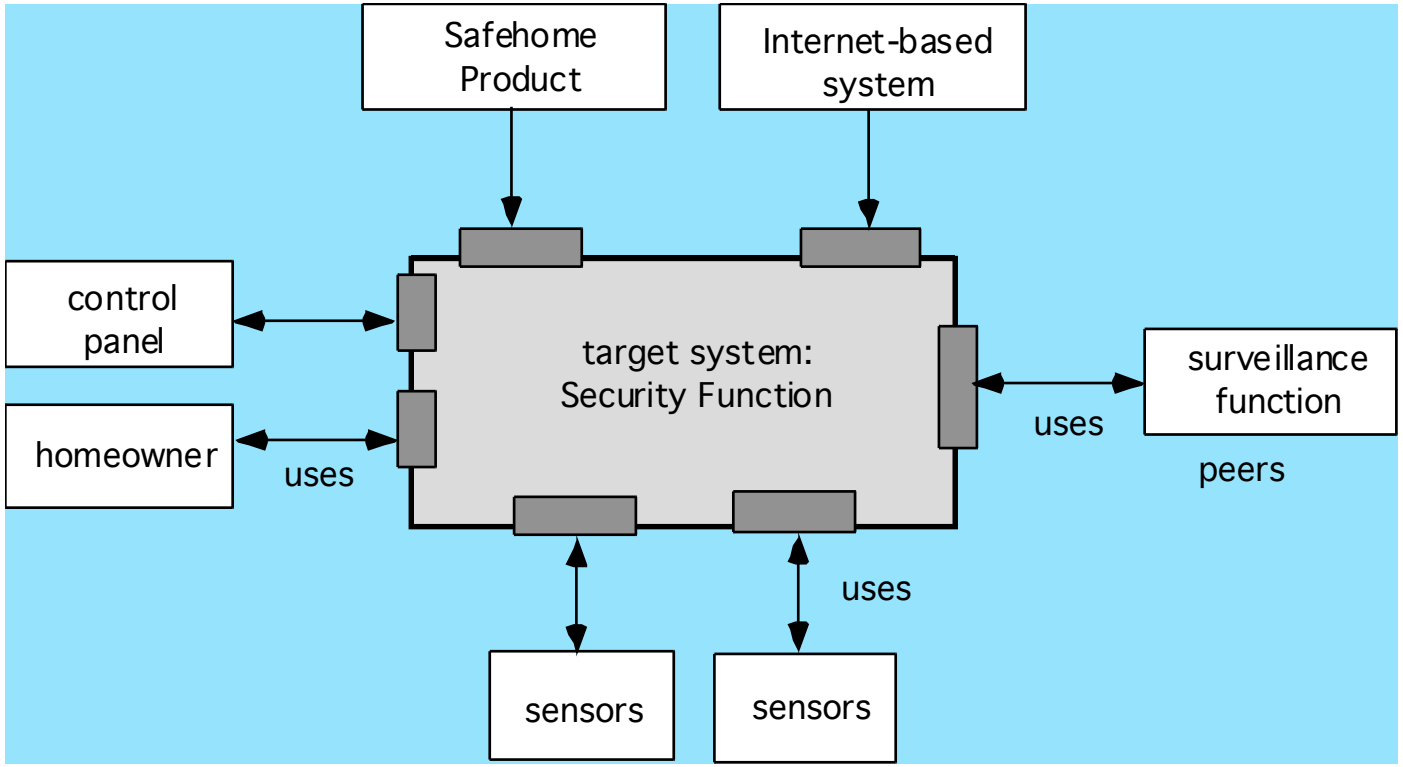
A graphical representation of the software to be built

An enabler for communication between all stakeholders involved in the software development

Stakeholders can use as a basis for mutual understanding and negotiation

Facilitates early design decisions

ARCHITECTURAL CONTEXT



ARCHITECTURAL CONSIDERATIONS

Economy	The best software is uncluttered easy to understand.
Visibility	Architectural decisions and the reasons for them should be obvious.
Spacing	No hidden dependencies.
Symmetry	System is consistent and balanced in its attributes.
Emergence	Emergent, self-organized behavior and control.
Documentation	Document all decisions as they are made.

ARCHITECTURE REVIEWS

- Assess the ability to meet the quality requirements
- Identify potential risks
- Economical in terms of effort and resources
- Often make use of experience-based reviews, prototype evaluation, and scenario reviews, and checklists

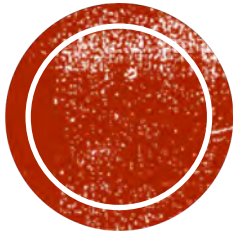
AGILITY AND ARCHITECTURE

To avoid rework, user stories are used to create and evolve an architectural model (walking skeleton) before coding

Hybrid models which allow software architects contributing users stories to the evolving storyboard

Well run agile projects include delivery of work products during each sprint

Reviewing code emerging from the sprint can be a useful form of architectural review

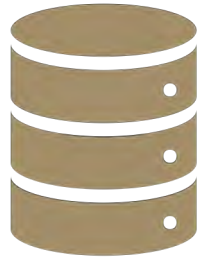


COMPONENT LEVEL DESIGN

WHAT IS A COMPONENT?

OO view

a component contains a set of collaborating classes



Conventional view

a component contains processing logic, data structures, and an interface

BASIC DESIGN PRINCIPLES

SRP - The Single Responsibility Principle

A class should have one, and only one, reason to change.

OCP - The Open Closed Principle

You should be able to extend a classes behavior, without modifying it.

LSP - The Liskov Substitution Principle

Derived classes must be substitutable for their base classes.

DIP - The Dependency Inversion Principle

Depend on abstractions, not on concretions.

ISP - The Interface Segregation Principle

Make fine grained interfaces that are client specific

DESIGN GUIDELINES

Components

Simple and easy to understand

Interfaces

Provide important information about communication and collaboration

Dependencies

Model dependencies from left to right

Inheritance

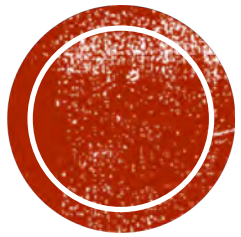
Model inheritance from bottom (derived classes) to top (base classes)

Reusability

Design reusable components

Reusable components - advantages

Adaptable	Adapt itself efficiently and fast to changed circumstances
Consistency	Same output for given input every time.
Extensibility	Easy to extend components
Fast	Faster development
Modularity	Small easy to use modules
Simple	Low complexity
Stability	Tested and stable



USER INTERFACE DESIGN

INTERFACE DESIGN

Easy to learn?

Easy to use?

Easy to understand?

TYPICAL DESIGN ERRORS

- Lack of consistency
- Too much memorization
- No guidance or help
- No context sensitivity
- Poor response time
- Complexity
- Not easy to use

GOLDEN RULES

Place the user in control

Reduce the user's memory load

Make the interface consistent

USER INTERFACE DESIGN MODELS

User model — a profile of all end users of the system

Design model — a design realization of the user model

Mental model — the user's mental image of what the interface is

Implementation model — the interface look and feel

INTERFACE ANALYSIS

- Understand the end-users who will interact with the system
- the tasks they must perform to do their work
- the content that is presented as part of the interface
- the environment in which these tasks will be conducted

USER ANALYSIS

- Who will be using the software?
- How they will use?
- Where they will use?
- Training
- Documentation
- Help and support

INTERFACE DESIGN STEPS

Define interface objects and actions (operations)

Define events (user actions)

Depict each interface state (look and feel)

Use consistent theme

Common design issues

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

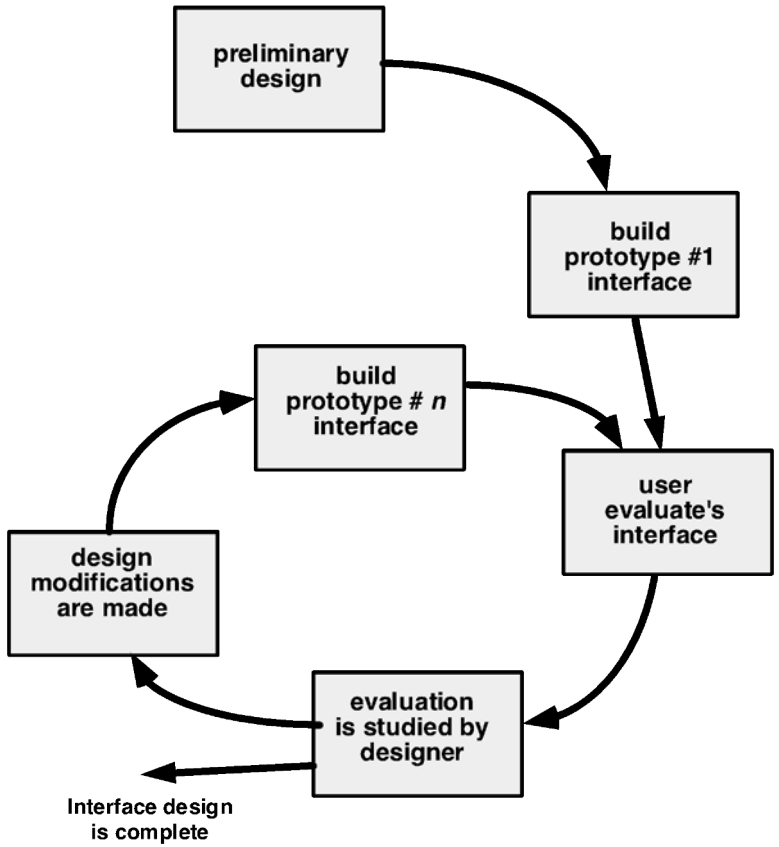
INTERFACE DESIGN PRINCIPLES

Usability	I can do it on my own and understand
Accessibility	Interface is accessible to all intended users
Anticipation	What user will do next?
Communication	Keep user informed, meaningful messages, where they are?
Consistency	Consistent look and feel (e.g., color, shape, layout)
Efficiency	The design should optimize the user's work efficiency

AESTHETIC DESIGN

Don’t	Don’t be afraid of white space
Emphasize	Emphasize content
Organize	Organize layout elements from top-left to bottom right
Group	Group navigation, content, and function geographically within the page
Don’t extend	Don’t extend your real estate with the scrolling bar
Consider	Consider resolution, window size, and device user will use

DESIGN EVALUATION CYCLE



REFERENCE

- Roger Pressman, Software Engineering: A Practitioner's Approach, 8th edition, McGraw Hill, ISBN 0078022126