Design Engineering

Software Engineering

DESIGN
REQUIREMENT
DEVELOPMENT
SOFTWARE ENGINEERING
PRODUCT RELEASE
TESTING
EVALUATION

Design Concepts

Design is where **customer requirements, business needs, and technical considerations** all come together in the formulation of a product or system

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

Design Concepts

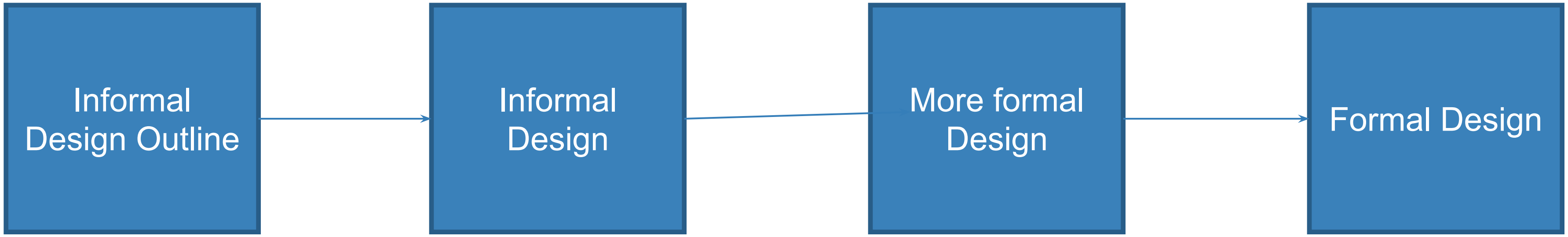
Goal of design engineering is to produce a model or representation that depict:

Firmness – Program should not have any bug that inhibits its functions.

Commodity – Suitable to its intended use.

Delight - Pleasurable to use

Design Concepts



Informal Design Outline

Informal Design

More formal Design

Formal Design

Design Within the Context of Software Engineering

- Sets the stage for Construction

- The design model provides detail about software **data structures, architecture, interfaces, and components** that are necessary to implement the system

- Software design model consists of 4 designs:

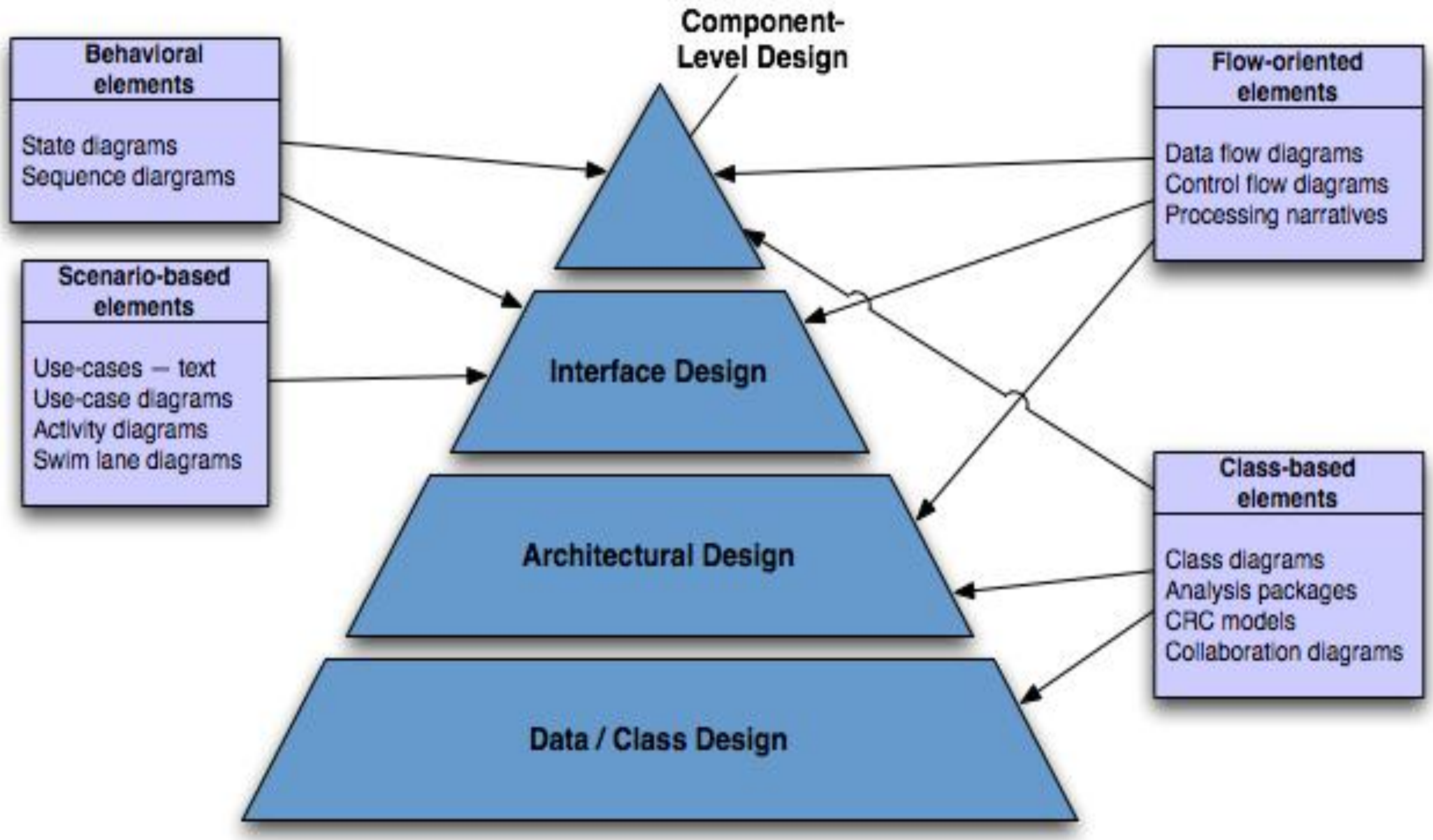
  Data/class Design

  Architectural Design

  Interface Design

  Component Design

Translating Analysis → Design



Data/class design

Created by transforming the analysis model **class-based elements** into **classes and data structures** required to implement the software

Architectural design

Defines the **relationships** among the major structural elements of the software, it is derived from the class-based elements and flow-oriented elements of the analysis model

Design Pattern

Architectural Styles

Interface design

Describes how the software elements, hardware elements, and end-users **communicate with one another**, it is derived from the analysis model scenario-based elements, flow-oriented elements, and behavioral elements

Component-level design

Created by transforming the structural elements defined by the software architecture into a procedural description of the **software components** using information obtained from the analysis model class-based elements, flow-oriented elements, and behavioral elements

Why Design is so Important?

- It is place where **quality** is fostered.
- It provides us with representation of software that can be assessed for quality.
- Only way that can accurately translate a customer’s requirements into a finished software product.
- It serves as foundation for all software engineering activities.
- Without design difficult to assess:

  Risk , Test and Quality

Design Process

Guide for the evaluation of Good Design

Must implement all of the **explicit requirements** contained in analysis model and **implicit requirements** desired by customer.

Must be **readable and understandable** by implementers, testers and maintainers alike.

Should provide complete picture of the software, addressing functional, behavioral and data domains **from an implementation perspective.**

What criteria would help to evaluate design quality?

Quality Guidelines and Technical Reviews

1. A design should exhibit an architecture that:

- has been created using recognizable architectural styles and patterns;
- is composed of components that exhibit good design characteristics;
- can be designed in evolutionary fashion.

What criteria would help to evaluate design quality?

2. A design should be modular.
3. A design should contain distinct representation of data, architecture, interfaces, and components.
4. A design should lead to data structures that are appropriate for classes to be implemented and are drawn from recognizable patterns.
5. A design should lead to components that exhibit independent functional characteristics.

What criteria would help to evaluate design quality?

6. A design should lead to interfaces that reduce the complexity of connections between components and with external environment.
7. A design should be derived using a repeatable method that is driven by information obtained during software requirements analysis.
8. A design should be represented using a notation that effectively communicates its meaning.

Software Quality Attributes (FURPS)

Functionality

Usability

Reliability

Performance

Supportability

Functionality

Is assessed by evaluating the ***feature set*** *and* ***capabilities*** *of the program, the generality of the functions that are delivered, and the security of the overall system.*

Usability

Is assessed by considering ***human factors, overall aesthetics, consistency, and documentation****.*

Reliability

Is evaluated by measuring the frequency and severity of failure, the accuracy of output results, the mean-time-to-failure (MTTF), the ability to recover from failure, and the predictability of the program.

Performance

Is measured by considering processing speed, response time, resource consumption, throughput, and efficiency.

Supportability

Combines the ability to extend the program (extensibility), adaptability, serviceability, maintainability , testability, compatibility and configurability (the ability to organize and control elements of the software configuration)

Design Concepts

Design concepts provide the necessary framework for "to get the thing on right way".

Abstraction

At the highest level of abstraction – a solution is stated in broad terms

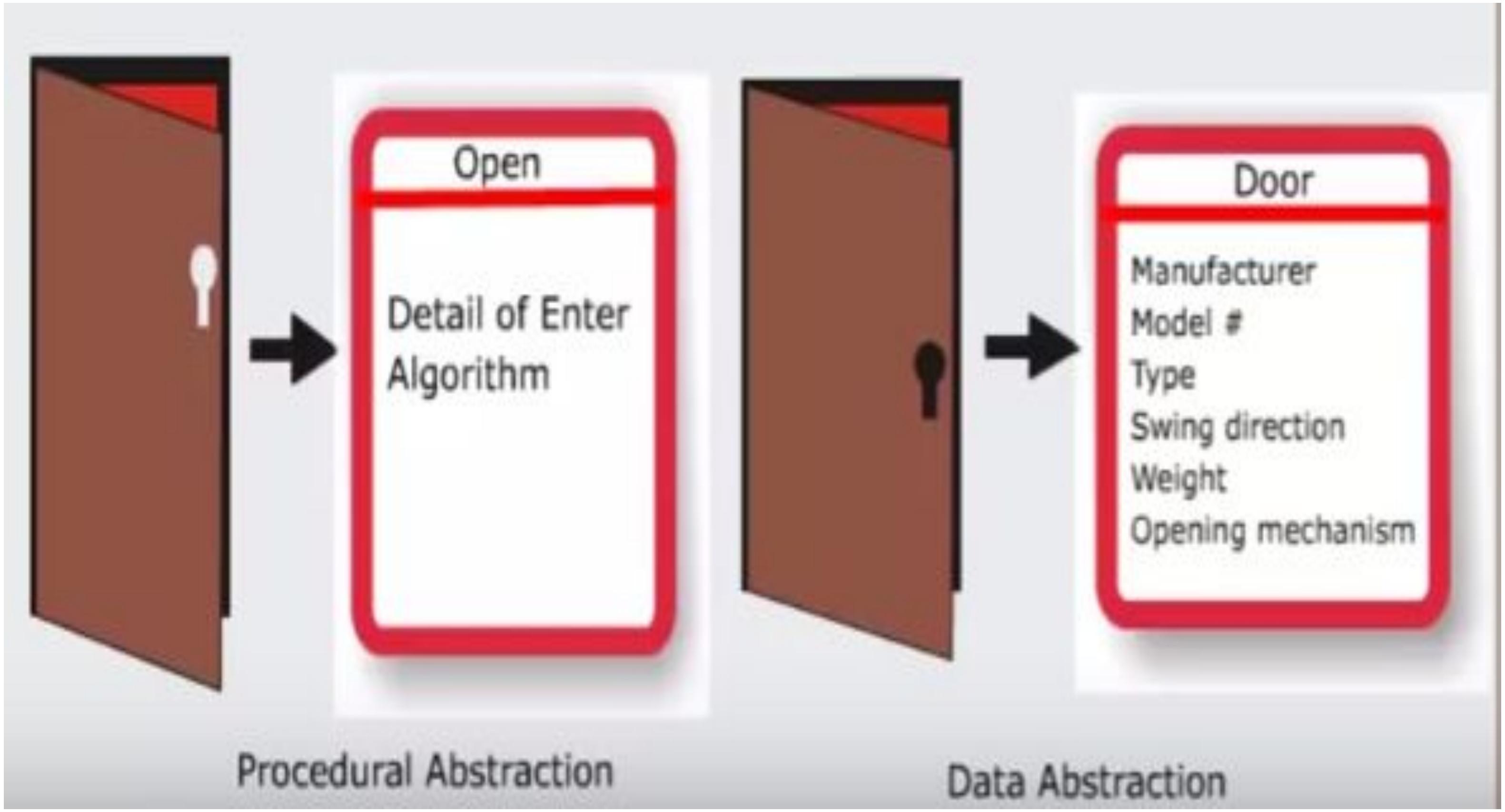
At lower level of abstraction – a more detailed description of the solution is provided.

Two types of abstraction:

***Procedural abstraction**:* Sequence of instructions that have a specific and limited function.

Data abstraction: Collection of data that describes a data object.

Abstraction



Architecture

Is the structure or **organization of program components**, the manner in which these **components interact and the data** that are used by the components.

Properties (Shaw and Garlan)

Structural Properties – Components of the System
Extra Functional Properties – Non Functional Requirements
Families of Related Systems – Repeatable Patterns

Architecture - Models

Structural Model- represent architecture as an organized collection of components

Framework model – Increase level of design abstraction by identifying repeatable architectural design framework.

Dynamic model – address behavior of the program architecture and indicating how system or structure configuration may change as a function of external event.

Process Model – focus on design of the business or technical process that the system must accommodate.

Functional models – used to represent the functional hierarchy of a system.

ADL – Architectural Description Languages

Patterns

Describes a design structure that solves a particular design problem within a specific context.

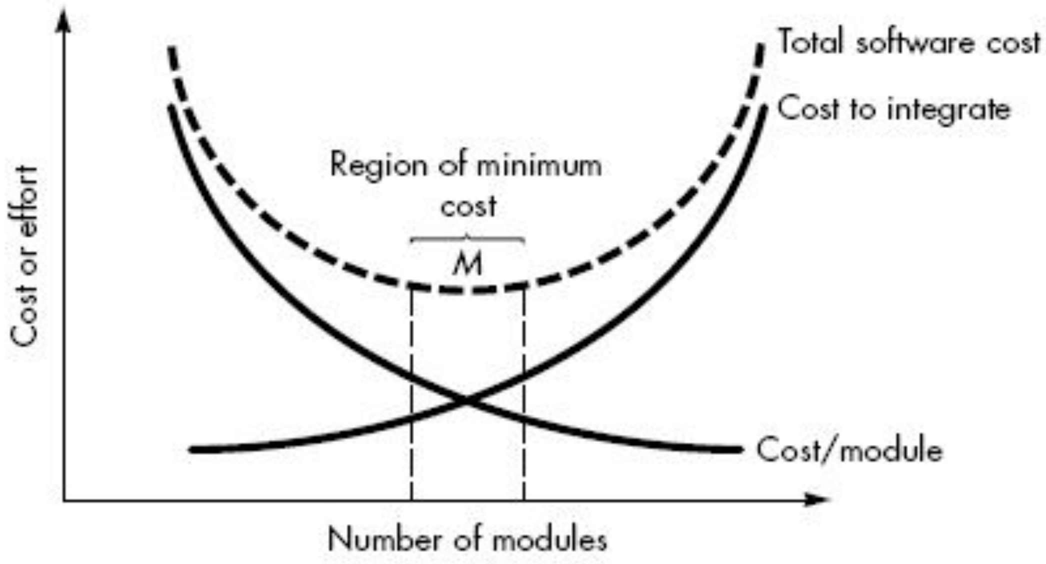
Should provide description that enables designer to determine:

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

Separation of Concerns

Modularity

Software is divided into separately named and addressable components (Modules), that are integrated to satisfy problem requirements.



Modularity

Information Hiding

Abstraction

Functional Independence – Qualitative Criteria

Cohesion

An indication of relative functional strength of a module

Coupling

An indication of the relative interdependence among modules

Refinement

Top down strategy

Process of Elaboration : Abstraction → Detailed Description

Refactoring

Is a reorganization technique that simplifies the design of a component without changing its function or behaviour.

Refactoring is the process of changing a software system in such a way that it does not alter the external behaviour of the code yet improves its internal structure.

Object Oriented Design Concepts

Object Oriented Paradigm is widely used in software engineering.

Design Classes

- Details to enable the classes to be implemented
- Analysis Classes → Design Classes

Five Types of Design Classes

User Interface Classes

All abstraction that are necessary for HCI

Business Domain Classes

The 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 manage the business domain classes.

Persistent Classes

Represent data stores

System Classes

System Flow Representations

Characteristics of Design Classes

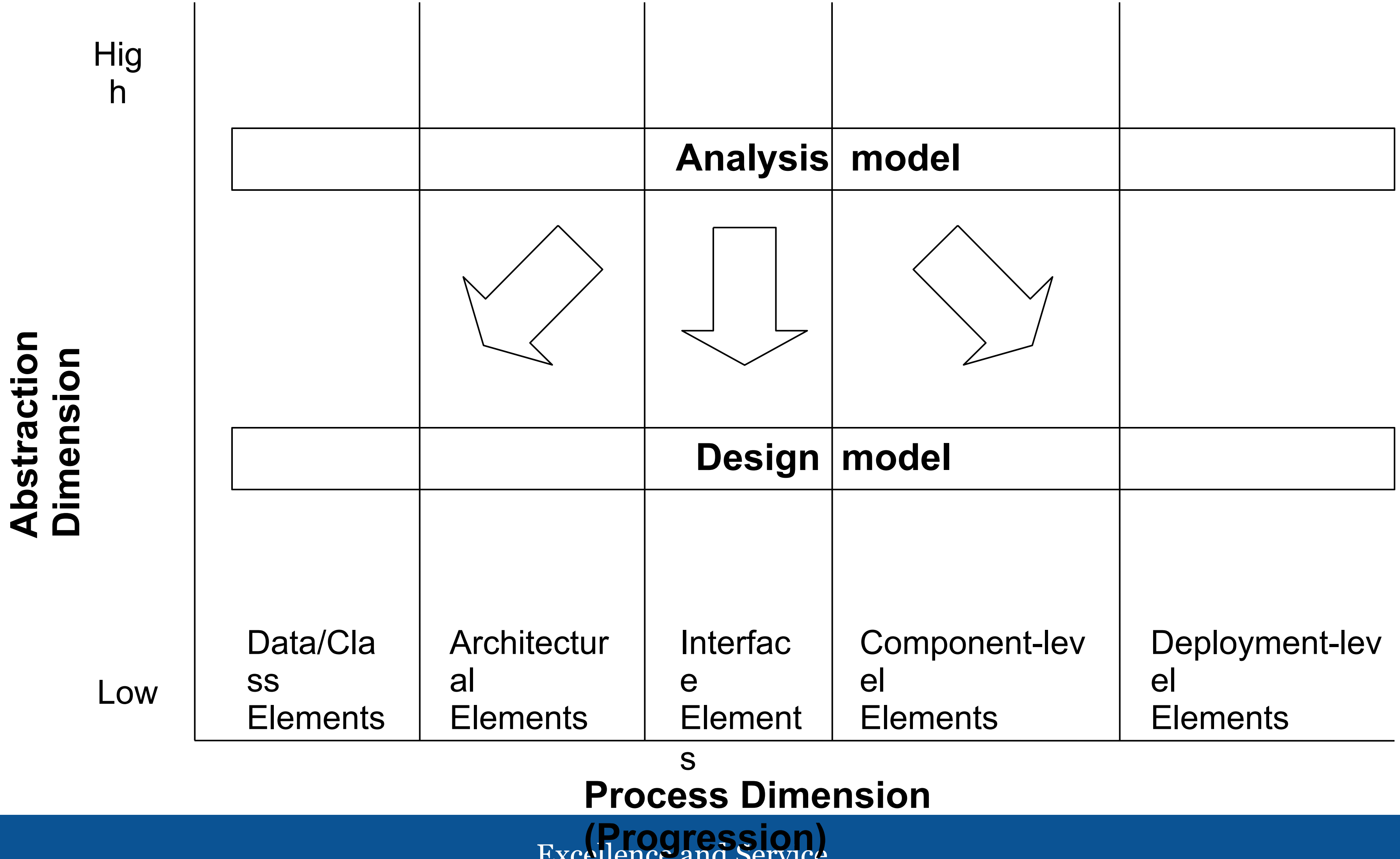
Complete and Sufficient

Primitiveness

High Cohesion

Low Coupling

Dimensions of the Design Model



High

Analysis model

Design model

Low

Data/Class Elements

Architectural Elements

Interface Elements

Component-level Elements

Deployment-level Elements

Abstraction Dimension

Process Dimension (Progression)

Design Model

The design model can be viewed in two different dimensions

(Horizontally) The process dimension indicates the evolution of the parts of the design model as each design task is executed

(Vertically) The abstraction dimension represents the level of detail as each element of the analysis model is transformed into the design model and then iteratively refined

Design Model

Elements of the design model use many of the same UML diagrams used in the analysis model

- The diagrams are refined and elaborated as part of the design
- More implementation-specific detail is provided
- Emphasis is placed on
  - Architectural structure and style
  - Interfaces between components and the outside world
  - Components that reside within the architecture

Design Elements

Data/class design

- Creates a model of data and objects that is represented at a high level of abstraction

Architectural design

- Depicts the overall layout of the software

Interface design

- Tells how information flows into and out of the system and how it is communicated among the components defined as part of the architecture
- Includes the user interface, external interfaces, and internal interfaces

Component-level design elements

- Describes the internal detail of each software component by way of data structure definitions, algorithms, and interface specifications

Deployment-level design elements

- Indicates how software functionality and subsystems will be allocated within the physical computing environment that will support the software

User Interface Design

Effective Communication Medium between Human and Computer

Software Engineer designs the User Interface by applying an iterative process

If the software is difficult to use, you won’t like it…

Interface mechanisms should be well designed

The Golden Rules

Theo Mandel Coins Three Golden Rules

1. Place the user in control
2. Reduce the user’s memory load
3. Make the interface consistent

Place the user in Control

1. Define interaction modes in a way that does not force a user into unnecessary or undesired actions
2. Provide for flexible interaction
3. Allow user interaction to be interruptible and undoable
4. Streamline interaction as skill levels advance and allow the interaction to be customized
5. Hide the technical internals from the casual user
6. Design for direct interaction with objects that appear on the screen

Reduce the User’s Memory Load

1. Reduce demand on short-term memory

2. Establish meaningful defaults

3. Describe shortcuts that are intuitive

4. The visual layout of the interface should be based on a real world metaphor

5. Disclose information in a progressive fashion

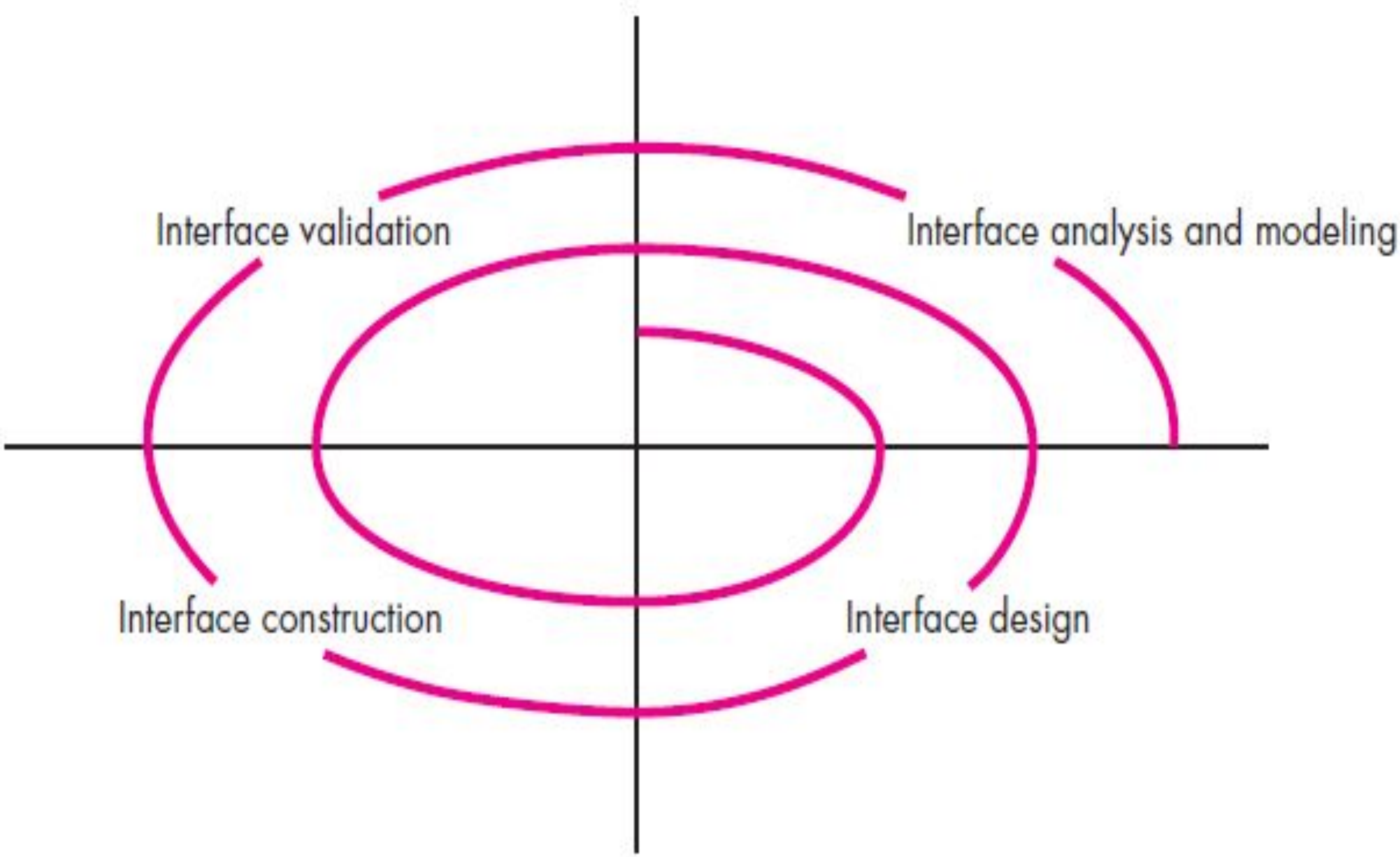
Make the Interface Consistent

Allow the user to put the current task into a meaningful context

Maintain consistency across a family of applications

If past interactive models have created user expectations, do not make changes unless there is a compelling reason to do so

User Interface Design – A Spiral Process



Spiral Process

Interface analysis (user, task, and environment analysis)

1. Focuses on the profile of the users who will interact with the system
2. Concentrates on users, tasks, content and work environment
3. Studies different models of system function (as perceived from the outside)
4. Defines the human- and computer-oriented tasks that are required to achieve system function

Interface design

5. Defines a set of interface objects and actions (and their screen representations) that enable a user to perform all defined tasks in a manner that meets every usability goal defined for the system

Interface construction

6. Begins with a prototype that enables usage scenarios to be evaluated
7. Continues with development tools to complete the construction

Interface validation

8. The ability of the interface to implement every user task correctly, to accommodate all task variations, and to achieve all general user requirements
9. The degree to which the interface is easy to use and easy to learn
10. The users' acceptance of the interface as a useful tool in their work

User Interface Analysis and Design

Four different models come into play when a user interface is analyzed and designed

User profile model – Established by a human engineer or software engineer

User's mental model – Developed by the user when interacting with the application

Design model – Created by a software engineer

Implementation model – Created by the software implementers

The role of the **interface designer is to reconcile these differences** and derive a consistent representation of the interface

User Profile Model

Establishes the profile of the end-users of the system

Based on age, gender, physical abilities, education, cultural or ethnic background, motivation, goals, and personality

Considers syntactic knowledge of the user

The mechanics of interaction that are required to use the interface effectively

Considers semantic knowledge of the user

The underlying sense of the application; an understanding of the functions that are performed, the meaning of input and output, and the objectives of the system

Categories of Users

Novices

No syntactic knowledge of the system, little semantic knowledge of the application, only general computer usage

Knowledgeable, intermittent users

Reasonable semantic knowledge of the system, low recall of syntactic information to use the interface

Knowledgeable, frequent users

Good semantic and syntactic knowledge (i.e., power user), look for shortcuts and abbreviated modes of operation

User’s Mental Model

Often called the user's system perception

Consists of the image of the system that users carry in their heads

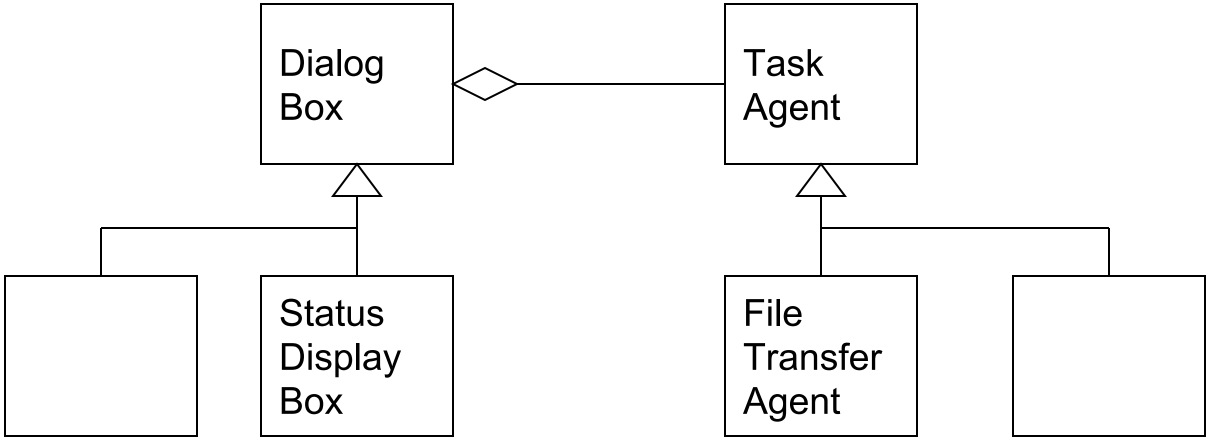
Accuracy of the description depends upon the user’s profile and overall familiarity with the software in the application domain

Design Model

Derived from the analysis model of the requirements

Incorporates data, architectural, interface, and procedural representations of the software

Constrained by information in the requirements specification that helps define the user of the system



Implementation Model

Consists of the look and feel of the interface combined with all supporting information (books, videos, help files) that describe system syntax and semantics

Strives to agree with the user's mental model; users then feel comfortable with the software and use it effectively

Serves as a translation of the design model by providing a realization of the information contained in the user profile model and the user’s mental model

Interface Analysis

User Analysis

- How do we learn what the user wants from UI?
- Mental Image of the UI can be collected by
- User Interviews, Sales Input, Marketing Input, Support Input

Task Analysis and Modelling

- Description of the Use case
- Task Elaboration, Object Elaboration and Workflow Analysis

Analysis of Display Content

- Format and Aesthetics of content displayed

Analysis of Work Environment

- Physical Environmental Factors

Interface Design Steps

Follow the Golden Rules

Parse the User Scenarios

- Find the Objects and Actions
- Designing the Screen Layout

User Interface Design Patterns

- Usage of design patterns (Calendar)

Design Issues

System Response Time – Length and Variability

User Help Facilities – User Manuals

Error Information Handling

Command Labelling

Thank You