CHITKARA UNIVERSITY
HIMACHAL PRADESH - NAAC Accredited

Object Oriented Software Engineering

LECTURE 9

Today's Outline:

Design Engineering:

- Design concepts and model
- Data design
- Architectural design
- Designing class based components
- User interface analysis and design
- Interface analysis and Interface design steps

What is it?

- Design is what almost every engineer wants to do. It is the place where creativity rules—where stakeholder requirements, business needs, and technical considerations all come together in the formulation of a product or system. Design creates a representation or model of the software, but unlike the requirements model, the design model provides detail about software architecture, data structures, interfaces, and components that are necessary to implement the system.

Who does it?

- Software engineers conduct each of the design tasks. Why is it important? Design allows you to model the system or product that is to be built. This model can be assessed for quality and improved before code is generated, tests are conducted, and end users become involved in large numbers. Design is the place where software quality is established.

What are the steps?

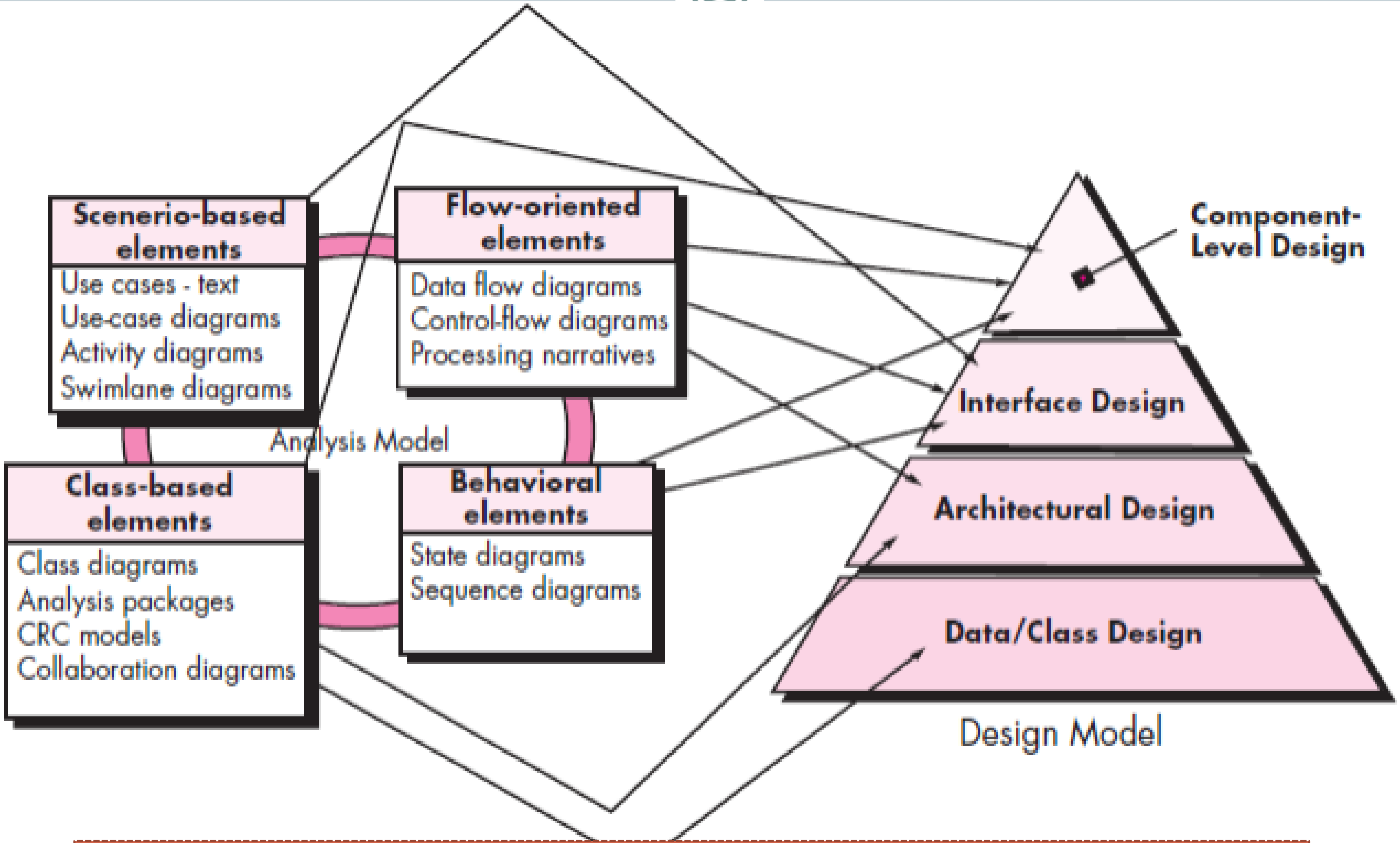
- Design depicts the software in a number of different ways. First, the architecture of the system or product must be represented. Then, the interfaces that connect the software to end users, to other systems and devices, and to its own constituent components are modeled. Finally, the software components that are used to construct the system are designed. Each of these views represents a different design action, but all must conform to a set of basic design concepts that guide software design work.

DESIGN WITHIN THE CONTEXT OF SOFTWARE ENGINEERING

- Software design sits at the technical kernel of software engineering and is applied regardless of the software process model that is used. Software design is the last software engineering action within the modeling activity and sets the stage for construction (code generation and testing).

CHITKARA UNIVERSITY
HIMACHAL PRADESH - NAAC Accredited

DESIGN WITHIN THE CONTEXT OF SOFTWARE ENGINEERING



Translating the requirements model into the design model

- Each of the elements of the requirements model provides information that is necessary to create the four design models required for a complete specification of design. The flow of information during software design is illustrated in previous figure. The requirements model, manifested by scenario-based, class-based, flow-oriented, and behavioral elements, feed the design task.

- The data/class design transforms class models into design class realizations and the requisite data structures required to implement the software. The objects and relationships defined in the CRC diagram and the detailed data content depicted by class attributes and other notation provide the basis for the data design action. Part of class design may occur in conjunction with the design of software architecture.
- The architectural design defines the relationship between major structural elements of the software, the architectural styles and design patterns that can be used to achieve the requirements defined for the system, and the constraints that affect the way in which architecture can be implemented.

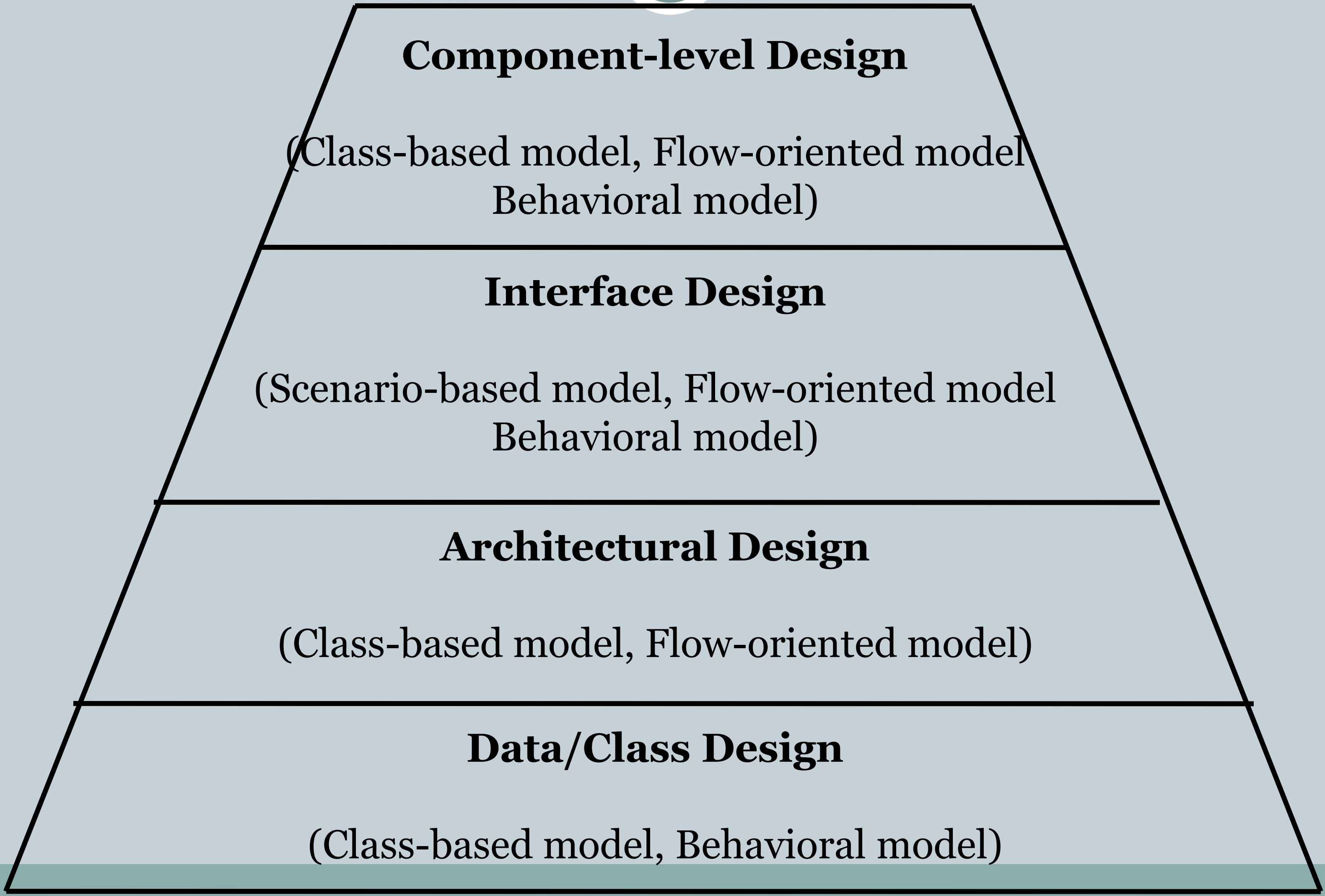
- The interface design describes how the software communicates with systems that interoperate with it, and with humans who use it. An interface implies a flow of information (e.g., data and/or control) and a specific type of behavior. Therefore, usage scenarios and behavioral models provide much of the information required for interface design.

- The component-level design transforms structural elements of the software architecture into a procedural description of software components. Information obtained from the class-based models, flow models, and behavioral models serve as the basis for component design. During design you make decisions that will ultimately affect the success of software construction and, as important, the ease with which software can be maintained.

Purpose of Design

- Design is where customer requirements, business needs, and technical considerations all come together in the formulation of a product or system
- The design model provides detail about the software data structures, architecture, interfaces, and components.
- The design model can be assessed for quality and be improved before code is generated and tests are conducted
  - Does the design contain errors, inconsistencies, or omissions?
  - Are there better design alternatives?
  - Can the design be implemented within the constraints, schedule, and cost that have been established?

From Analysis Model to Design Model (continued)


Component-level Design
(Class-based model, Flow-oriented model
Behavioral model)
Interface Design
(Scenario-based model, Flow-oriented model
Behavioral model)
Architectural Design
(Class-based model, Flow-oriented model)
Data/Class Design
(Class-based model, Behavioral model)

CHITKARA UNIVERSITY
HIMACHAL PRADESH - NAAC Accredited

Design Quality

- Quality’s Role
- The importance of design is quality
    - Provides representations of software that can be assessed for quality
    - Accurately translates a customer's requirements into a finished software product or system
    - Serves as the foundation for all software engineering activities that follow
- The quality of the design is assessed through a series of formal technical reviews or design walkthroughs

Goals of a Good Design

- The design must implement all of the explicit requirements contained in the analysis model
  - It must also accommodate all of the implicit requirements desired by the customer
- The design must be a readable and understandable guide for those who generate code, and for those who test and support the software
- The design should provide a complete picture of the software, addressing the data, functional, and behavioral domains from an implementation perspective

Design Quality Guidelines

1) A design should exhibit an <u>architecture</u> that
   a) Has been created using recognizable <u>architectural styles or patterns</u>
   b) Is composed of components that exhibit good design characteristics
   c) Can be implemented in an <u>evolutionary</u> fashion, thereby facilitating implementation and testing
2) A design should be <u>modular</u>; that is, the software should be logically partitioned into elements or subsystems
3) A design should contain <u>distinct representations</u> of data, architecture, interfaces, and components
4) A design should lead to <u>data structures</u> that are <u>appropriate</u> for the classes to be implemented and are drawn from recognizable data patterns

Quality Guidelines (continued)

5) A design should lead to <u>components</u> that exhibit <u>independent</u> functional characteristics
6) A design should lead to interfaces that <u>reduce the complexity of connections</u> between components and with the external environment
7) A design should be derived using a repeatable method that is <u>driven by</u> information obtained during software <u>requirements analysis</u>
8) A design should be represented using a <u>notation</u> that effectively communicates its meaning

"Quality isn't something you lay on top of subjects and objects like tinsel on a Christmas tree."

Design Concepts

Design Concepts

- DESIGN CONCEPTS:

Design concepts has evolved over the history of software engineering. Each concept provides the software designer with a foundation from which more sophisticated design methods can be applied. A brief overview of important software design concepts that span both traditional and object-oriented software development is given below.

- Abstraction
- When you consider a modular solution to any problem, many levels of abstraction can be posed. 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. 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. The name of a procedural abstraction implies these functions, but specific details are suppressed. A data abstraction is a named collection of data that describes a data object.

Design Concepts

- Architecture:
- Software architecture alludes to "the overall structure of the software and the ways in which that structure provides conceptual integrity for a system". In its simplest form, architecture is the structure or organization of program components (modules), the manner in which these components interact, and the structure of data that are used by the components. One goal of software design is to derive an architectural rendering of a system. A set of architectural patterns enables a software engineer to solve common design problems.
- Shaw and Garlan describe a set of properties as part of an architectural design: 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. For example, objects are packaged to encapsulate both data and the processing that manipulates the data and interact via the invocation of methods.

Design Concepts

- 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.

CHITKARA UNIVERSITY
HIMACHAL PRADESH - NAAC Accredited

Design Concepts

- Patterns:
- A pattern is a named nugget of insight which conveys the essence of a proven solution to a recurring problem within a certain context amidst competing concerns. Stated A design pattern describes a design structure that solves a particular design problem within a specific context and amid "forces" that may have an impact on the manner in which the pattern is applied and used. 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 (hence, saving design time) (3) whether the pattern can serve as a guide for developing a similar, but functionally or structurally different pattern.

CHITKARA UNIVERSITY
HIMACHAL PRADESH - NAAC Accredited

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. For two problems, p1 and p2, if the perceived complexity of p1 is greater than the perceived complexity of p2, it follows that the effort required to solve p1 is greater than the effort required to solve p2. As a general case, this result is intuitively obvious. It does take more time to solve a difficult problem. It also follows that the perceived complexity of two problems when they are combined is often greater than the sum of the perceived complexity when each is taken separately. This leads to a divide-and-conquer strategy—it's easier to solve a complex problem when you break it into manageable pieces. This has important implications with regard to software modularity.

Design Concepts

- 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. It has been stated that "modularity is the single attribute of software that allows a program to be intellectually manageable". The number of control paths, span of reference, number of variables, and overall complexity would make understanding close to impossible. In almost all instances, you should break the design into many modules, hoping to make understanding easier and, as a consequence, reduce the cost required to build the software.

CHITKARA UNIVERSITY
HIMACHAL PRADESH - NAAC Accredited

Design Concepts

- Information Hiding:
- The principle of information hiding suggests that modules be "characterized by design decisions that (each) hides from all others." In other words, modules should be specified and designed so that information (algorithms and data) contained within a module is inaccessible to other modules that have no need for such information. Hiding implies that effective modularity can be achieved by defining a set of independent modules that communicate with one another only that information necessary to achieve software function. Abstraction helps to define the procedural (or informational) entities that make up the software.

CHITKARA UNIVERSITY
HIMACHAL PRADESH - NAAC Accredited

Design Concepts

- Functional Independence:
- The concept of functional independence is a direct outgrowth of separation of concerns, modularity, and the concepts of abstraction and information hiding. Independent modules are easier to maintain (and test) because secondary effects caused by design or code modification are limited, error propagation is reduced, and reusable modules are possible.
- 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

- A cohesive module performs a single task, requiring little interaction with other components in other parts of a program. Stated simply, a cohesive module should (ideally) do just one thing. Although you should always strive for high cohesion (i.e., single-mindedness), it is often necessary and advisable to have a software component perform multiple functions.
- Coupling is an indication of interconnection among modules in a software structure. Coupling depends on the interface complexity between modules, the point at which entry or reference is made to a module, and what data pass across the interface. In software design, you should strive for the lowest possible coupling. Simple connectivity among modules results in software that is easier to understand and less prone to a "ripple effect", caused when errors occur at one location and propagate throughout a system.

CHITKARA UNIVERSITY
HIMACHAL PRADESH - NAAC Accredited

Design Concepts

- Refinement:
- Stepwise refinement is a top-down design strategy. . 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 is actually a process of elaboration begins with a statement of function (or description of information) that is defined at a high level of abstraction. You then elaborate on the original statement, providing more and more detail as each successive refinement (elaboration) occurs. Refinement helps you to reveal low-level details as design progresses.

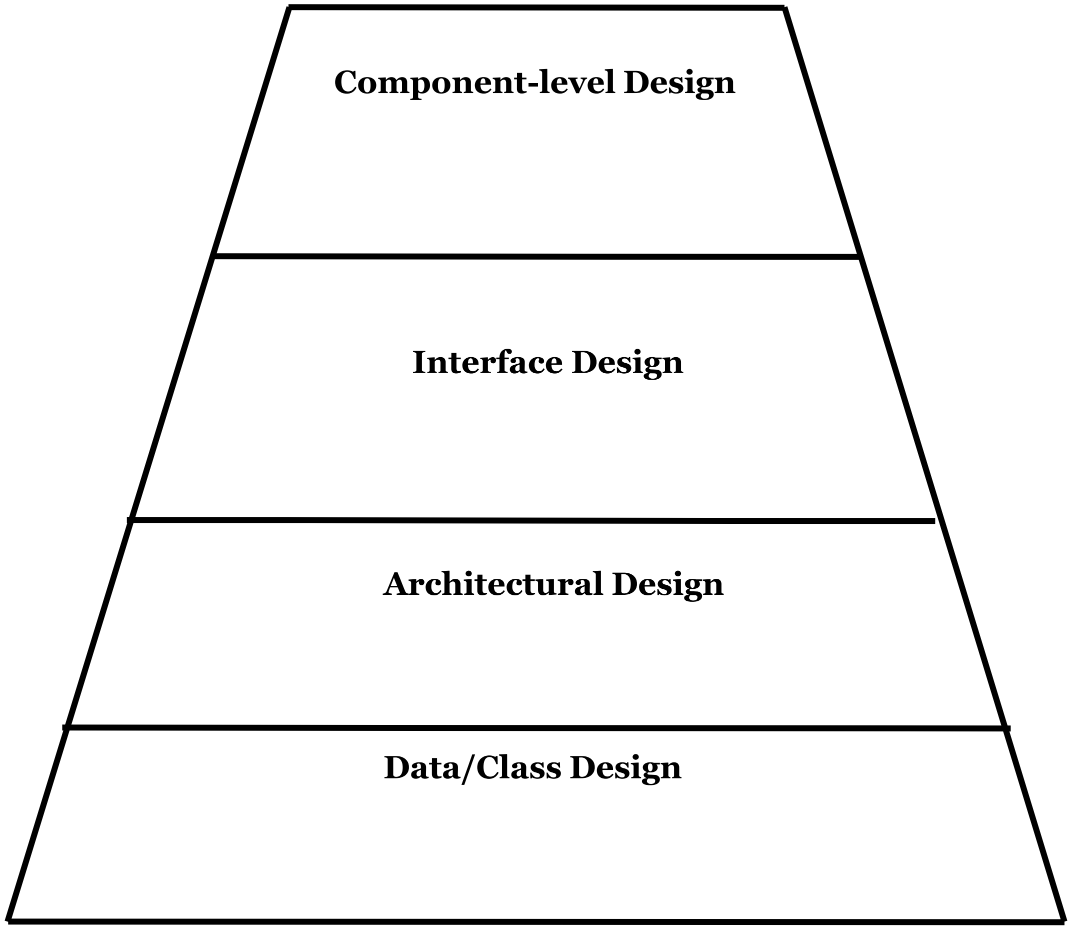
Design Concepts (continued)

- Stepwise refinement
  - Development of a program by successively refining levels of procedure detail
  - Complements abstraction, which enables a designer to specify procedure and data and yet suppress low-level details
- Refactoring
  - A reorganization technique that simplifies the design (or internal code structure) of a component without changing its function or external behavior
  - Removes redundancy, unused design elements, inefficient or unnecessary algorithms, poorly constructed or inappropriate data structures, or any other design failures
- Design classes
  - Refines the analysis classes by providing design detail that will enable the classes to be implemented
  - Creates a new set of design classes that implement a software infrastructure to support the business solution

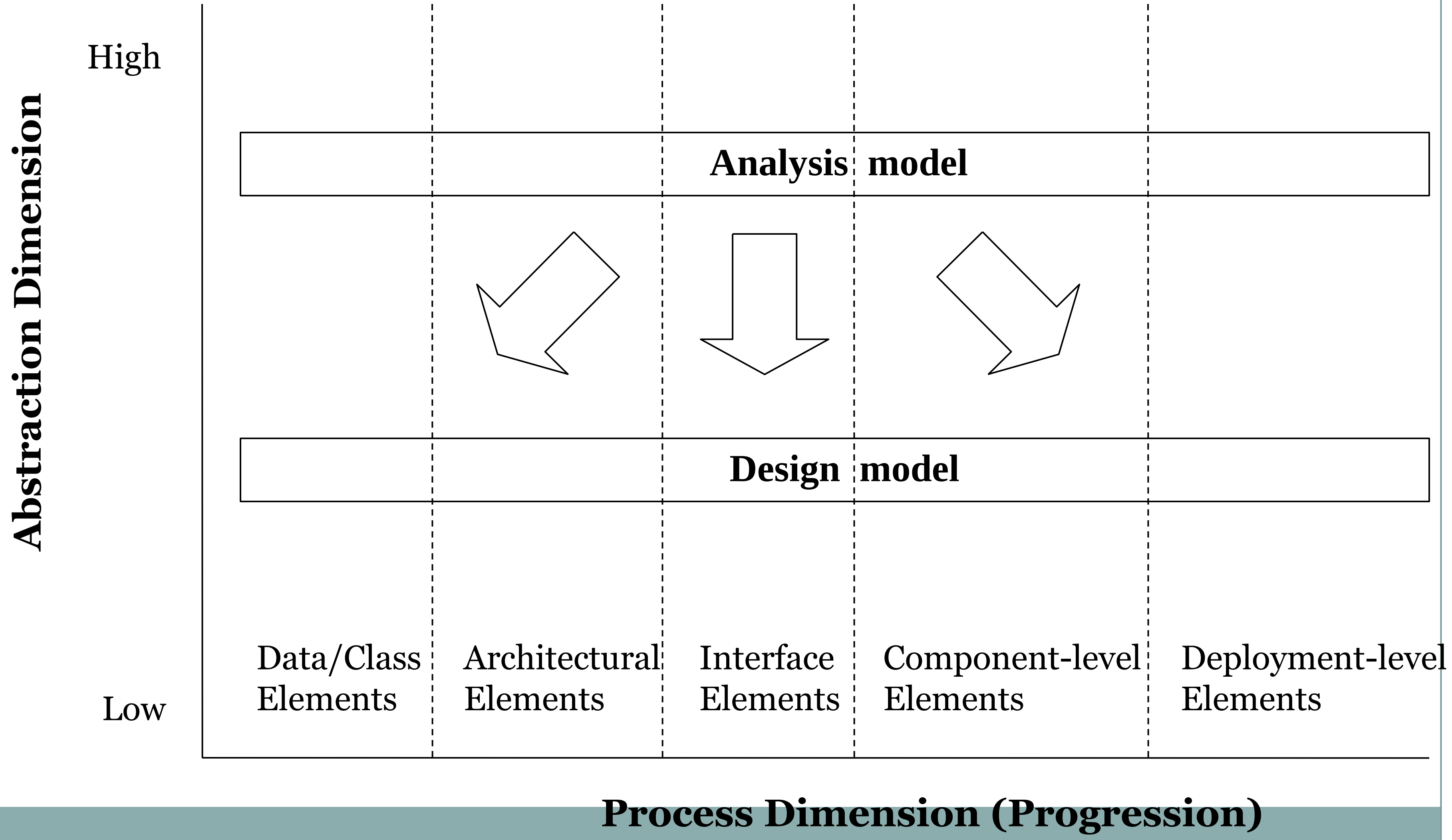
Characteristics of a Well-Formed Design Class

- Complete and sufficient
  - Contains the complete encapsulation of all attributes and methods that exist for the class
  - Contains only those methods that are sufficient to achieve the intent of the class
- Primitiveness
  - Each method of a class focuses on accomplishing one service for the class
- High cohesion
  - The class has a small, focused set of responsibilities and single-mindedly applies attributes and methods to implement those responsibilities
- Low coupling
  - Collaboration of the class with other classes is kept to an acceptable minimum
  - Each class should have limited knowledge of other classes in other subsystems

The Design Model



Dimensions of the Design Model



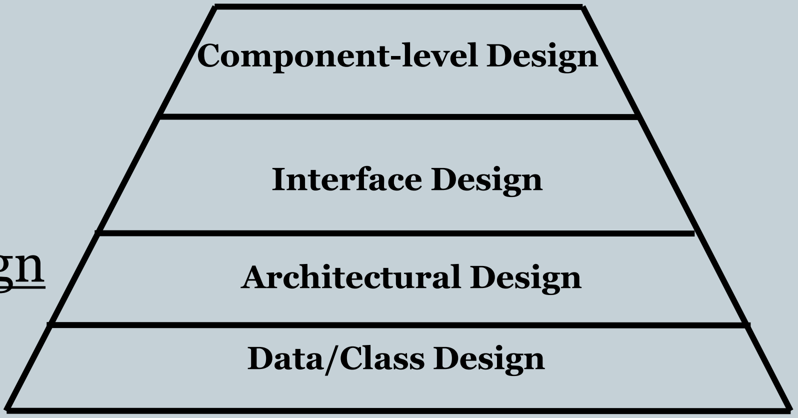
Introduction

- 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

(More on next slide)

Introduction (continued)

- Design model elements are <u>not always</u> developed in a <u>sequential</u> fashion
  - Preliminary architectural design sets the stage
  - It is followed by interface design and component-level design, which often occur <u>in parallel</u>
- The design model has the following layered elements
  - Data/class design
  - Architectural design
  - Interface design
  - Component-level design
- A fifth element that follows all of the others is <u>deployment-level design</u>



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

THANK YOU