Chapter 9

Design Engineering

- Introduction
- Design quality
- Design concepts
- The design model

Introduction

Five Notable Design Quotes

- "Questions about whether design is necessary or affordable are quite beside the point; design is inevitable. The alternative to good design is bad design, [rather than] no design at all." **Douglas Martin**
- "You can use an eraser on the drafting table or a sledge hammer on the construction site." **Frank Lloyd Wright**
- "The public is more familiar with bad design than good design. If is, in effect, conditioned to prefer bad design, because that is what it lives with; the new [design] becomes threatening, the old reassuring." **Paul Rand**
- "A common mistake that people make when trying to design something completely foolproof was to underestimate the ingenuity of complete fools." **Douglas Adams**
- "Every now and then go away, have a little relaxation, for when you come back to your work your judgment will be surer. Go some distance away because then the work appears smaller and more of it can be taken in at a glance and a lack of harmony and proportion is more readily seen." **Leonardo DaVinci**

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?

(More on next slide)

Purpose of Design (continued)

- A designer must practice diversification and convergence
  - The designer selects from design components, component solutions, and knowledge available through catalogs, textbooks, and experience
  - The designer then chooses the elements from this collection that meet the requirements defined by requirements engineering and analysis modeling
  - Convergence occurs as alternatives are considered and rejected until one particular configuration of components is chosen
- Software design is an iterative process through which requirements are translated into a blueprint for constructing the software
  - Design begins at a high level of abstraction that can be directly traced back to the data, functional, and behavioral requirements
  - As design iteration occurs, subsequent refinement leads to design representations at much lower levels of abstraction

From Analysis Model to Design Model

- Each element of the analysis model provides information that is necessary to create the four design models
  - The data/class design transforms analysis classes into design classes along with the data structures required to implement the software
  - The architectural design defines the relationship between major structural elements of the software; architectural styles and design patterns help achieve the requirements defined for the system
  - The interface design describes how the software communicates with systems that interoperate with it and with humans that use it
  - The component-level design transforms structural elements of the software architecture into a procedural description of software components

(More on next slide)

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)

Task Set for Software Design

1) Examine the information domain model and design appropriate data structures for data objects and their attributes
2) Using the analysis model, select an architectural style (and design patterns) that are appropriate for the software
3) Partition the analysis model into design subsystems and allocate these subsystems within the architecture
   a) Design the subsystem interfaces
   b) Allocate analysis classes or functions to each subsystem
4) Create a set of design classes or components
   a) Translate each analysis class description into a design class
   b) Check each design class against design criteria; consider inheritance issues
   c) Define methods associated with each design class
   d) Evaluate and select design patterns for a design class or subsystem

(More on next slide)

Task Set for Software Design (continued)

5) <u>Design</u> any interface required with external systems or devices
6) <u>Design</u> the user interface
7) <u>Conduct</u> component-level design
   a) Specify all algorithms at a relatively low level of abstraction
   b) Refine the interface of each component
   c) Define component-level data structures
   d) Review each component and correct all errors uncovered
8) <u>Develop</u> a deployment model
   - Show a physical layout of the system, revealing which components will be located where in the physical computing environment

Design Quality

Quality's Role

- The importance of design is quality
- Design is the place where quality is fostered
  - 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
- Without design, we risk building an unstable system that
  - Will fail when small changes are made
  - May be difficult to test
  - Cannot be assessed for quality later in the software process when time is short and most of the budget has been spent
- 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

"Writing a clever piece of code that works is one thing; designing something that can support a long-lasting business is quite another."

Design Quality Guidelines

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

(more on next slide)

Quality Guidelines (continued)

5) A design should lead to components that exhibit independent functional characteristics
6) A design should lead to interfaces that reduce the complexity of connections between components and with the 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

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

Design Concepts

Design Concepts

- Abstraction
  - Procedural abstraction – a sequence of instructions that have a specific and limited function
  - Data abstraction – a named collection of data that describes a data object
- Architecture
  - The overall structure of the software and the ways in which the structure provides conceptual integrity for a system
  - Consists of components, connectors, and the relationship between them
- Patterns
  - A design structure that solves a particular design problem within a specific context
  - It provides a description that enables a designer to determine whether the pattern is applicable, whether the pattern can be reused, and whether the pattern can serve as a guide for developing similar patterns

(more on next slide)

Design Concepts (continued)

- Modularity
  - Separately named and addressable components (i.e., modules) that are integrated to satisfy requirements (divide and conquer principle)
  - Makes software intellectually manageable so as to grasp the control paths, span of reference, number of variables, and overall complexity
- Information hiding
  - The designing of modules so that the algorithms and local data contained within them are inaccessible to other modules
  - This enforces access constraints to both procedural (i.e., implementation) detail and local data structures
- Functional independence
  - Modules that have a "single-minded" function and an aversion to excessive interaction with other modules
  - High cohesion – a module performs only a single task
  - Low coupling – a module has the lowest amount of connection needed with other modules

(more on next slide)

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

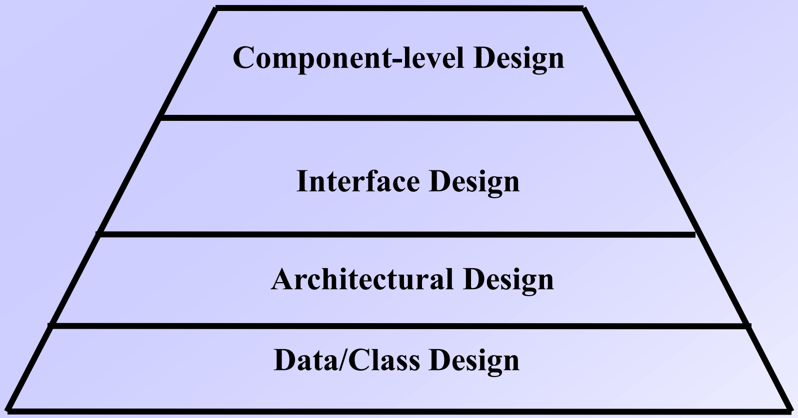
Types of Design Classes

- **User interface classes** – define all abstractions necessary for human-computer interaction (usually via metaphors of real-world objects)
- **Business domain classes** – refined from analysis classes; identify attributes and services (methods) that are required to implement some element of the business domain
- **Process classes** – implement 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 the outside world

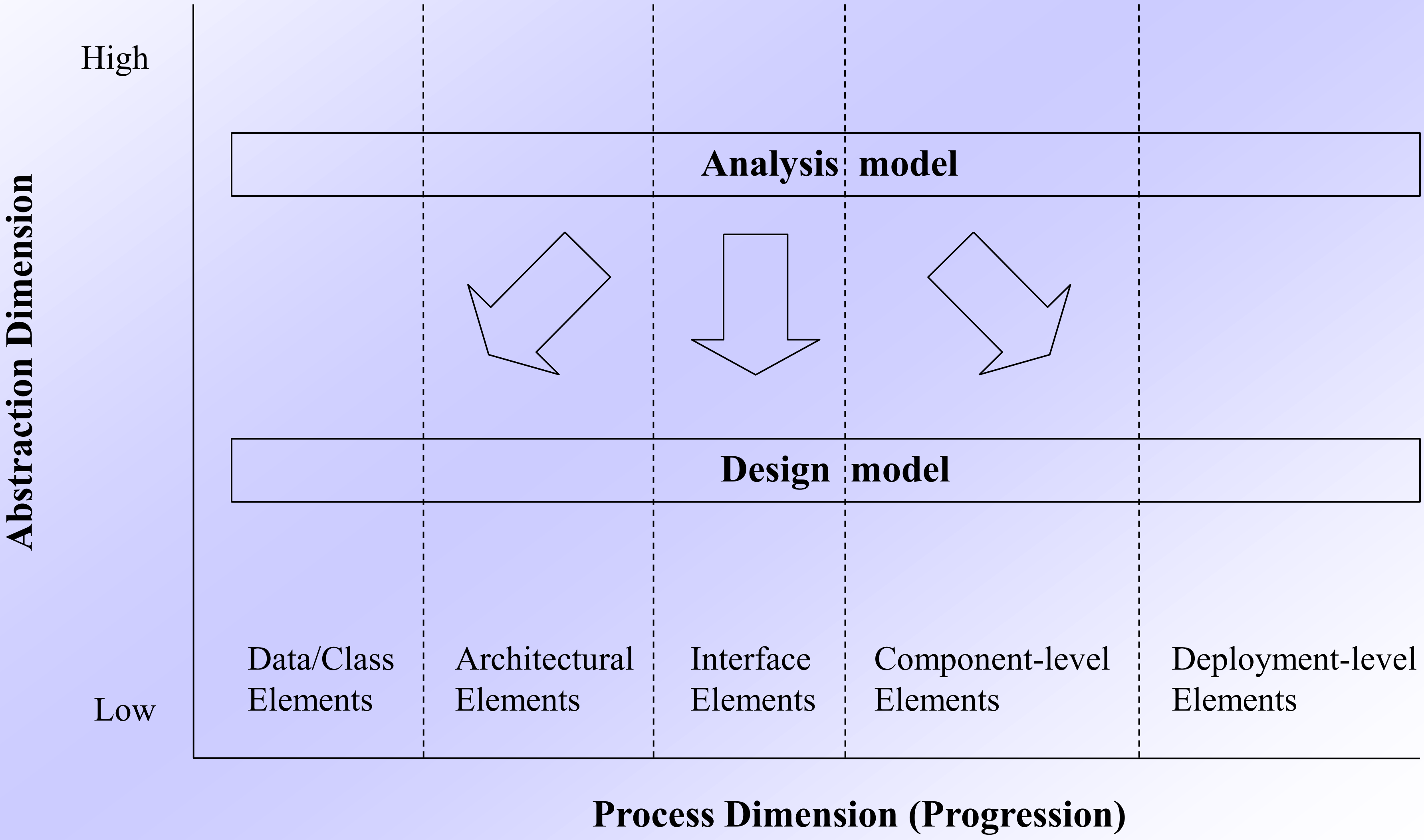
Characteristics of a Well-Formed Design Class

- 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



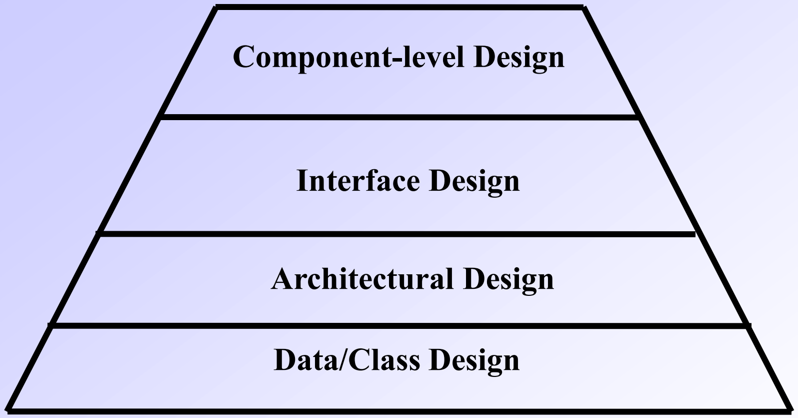
Introduction

- 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
- 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

(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

Pattern-based Software Design

- Mature engineering disciplines make use of thousands of design patterns for such things as buildings, highways, electrical circuits, factories, weapon systems, vehicles, and computers
- Design patterns also serve a purpose in software engineering
- Architectural patterns
  - Define the overall structure of software
  - Indicate the relationships among subsystems and software components
  - Define the rules for specifying relationships among software elements
- Design patterns
  - Address a specific element of the design such as an aggregation of components or solve some design problem, relationships among components, or the mechanisms for effecting inter-component communication
  - Consist of creational, structural, and behavioral patterns
- Coding patterns
  - Describe language-specific patterns that implement an algorithmic or data structure element of a component, a specific interface protocol, or a mechanism for communication among components