CS 123
Introduction to Software Engineering

06-A: Design Concepts and Principles

DISCS

SY 2013-2014

Quick Overview

- What is Software Design?
  - Describe HOW the system is supposed to do what it is supposed to do
  - Derive a solution which satisfies software requirements
- What is the input?
  - Requirements Analysis Specification
- What is the work output?
  - Design Specification
- Why is it important?
  - Building computer software is complex (typically a lot more complex than building a house) and needs some kind of blue-print – the design.

Design

- Some quotes on Design (in general):
  - "You can use an eraser on the drafting table or a sledge hammer on the construction site." Frank Lloyd Wright
  - "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
  - "A common mistake that people make when trying to design something completely fool proof 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

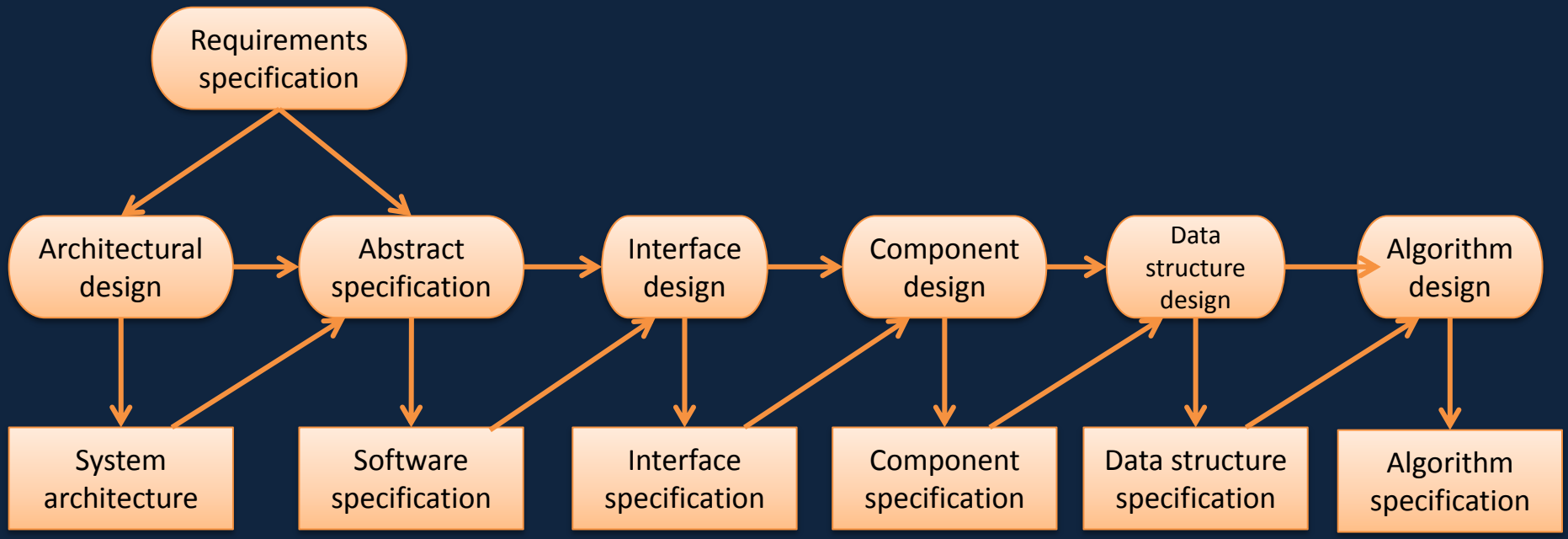
Software Design

- The designer's goal is to produce a model or representation of an entity that will later be built.
- Software Design sits at the core of software engineering
  - Applied regardless of the software process model that is used
- Begins after requirements have been analyzed and specified
- First of three technical activities required to build & verify the software
  - Design
  - Code generation
  - Testing

Stages of Design

- Problem understanding
  - Look at the problem from different angles to discover the design requirements.
- Identify one or more solutions
  - Evaluate possible solutions and choose the most appropriate depending on the designer's experience and available resources.
- Describe solution abstractions
  - Use graphical, formal or other descriptive notations to describe the components of the design.
- Repeat process for each identified abstraction until the design is expressed in primitive terms.

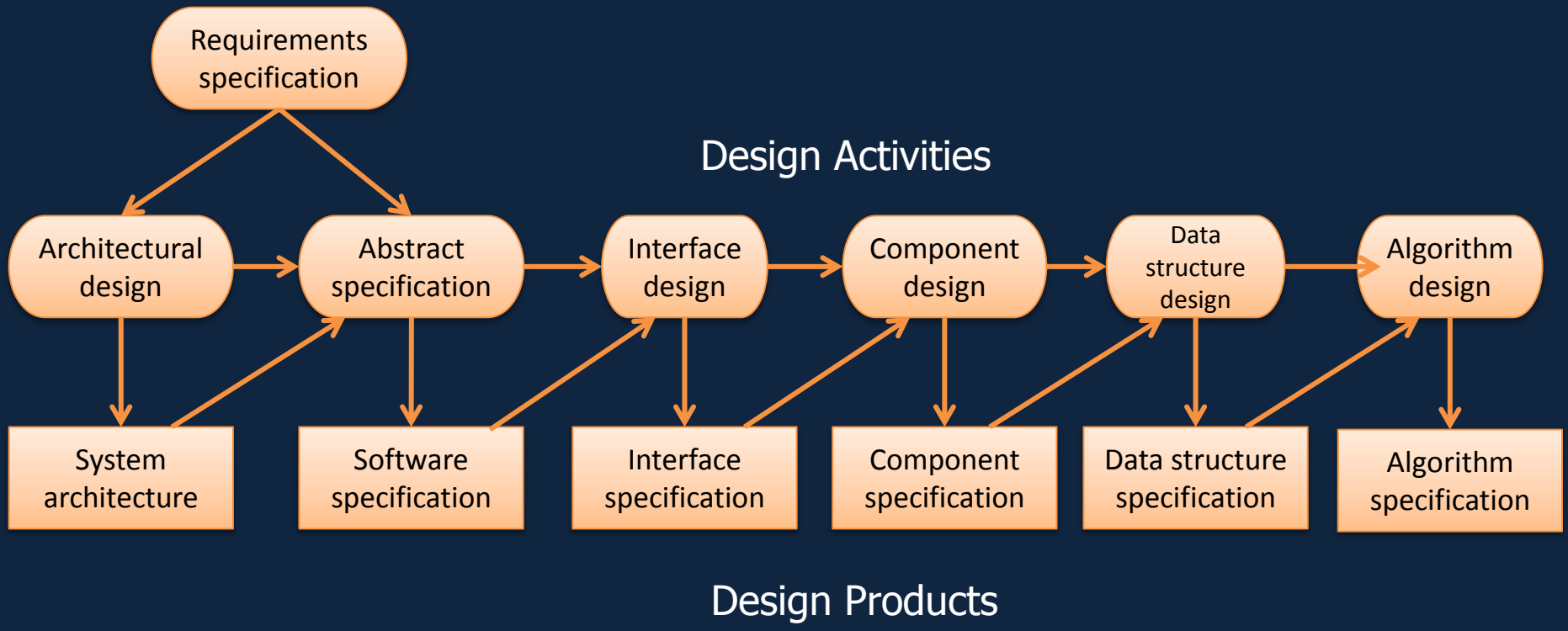
Phases in the Design Process



Architecture design – identify subsystems

Abstract specification – specify subsystems

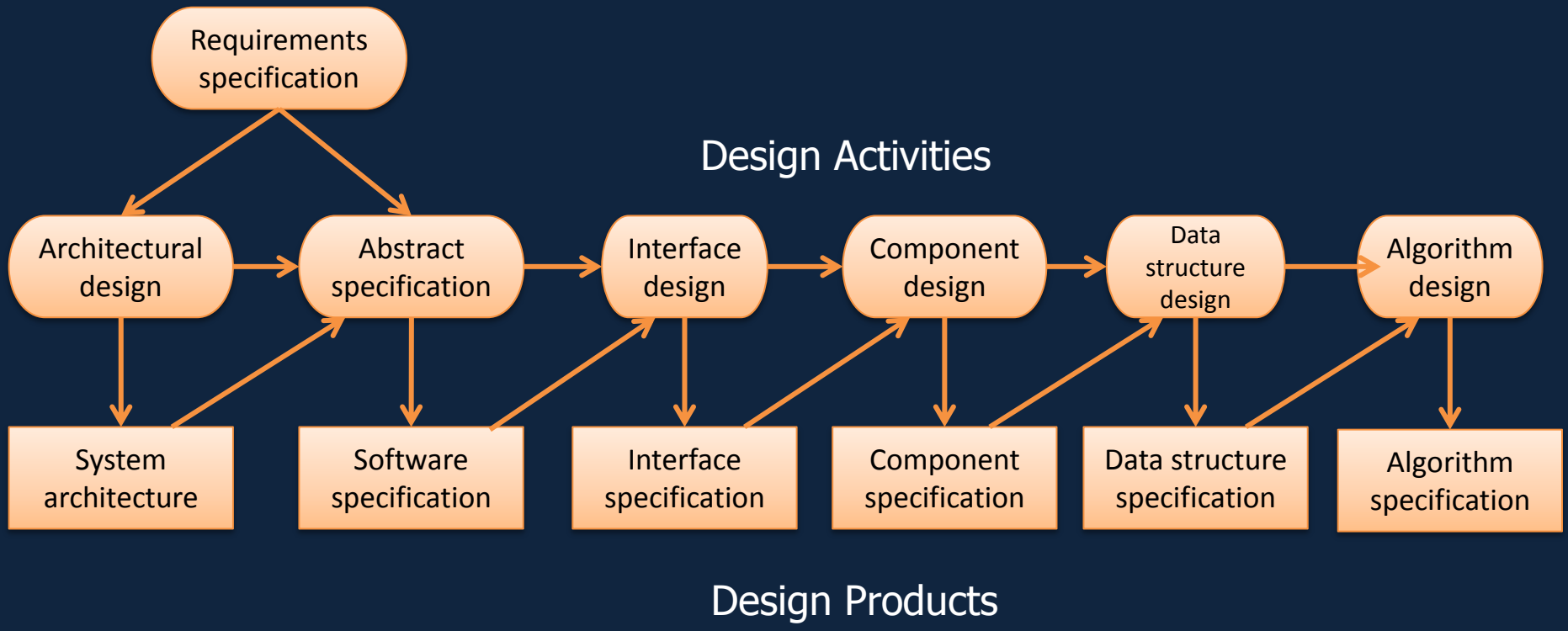
Phases in the Design Process



Interface design – describe subsystem interfaces

Component design – decompose subsystems into components

Phases in the Design Process



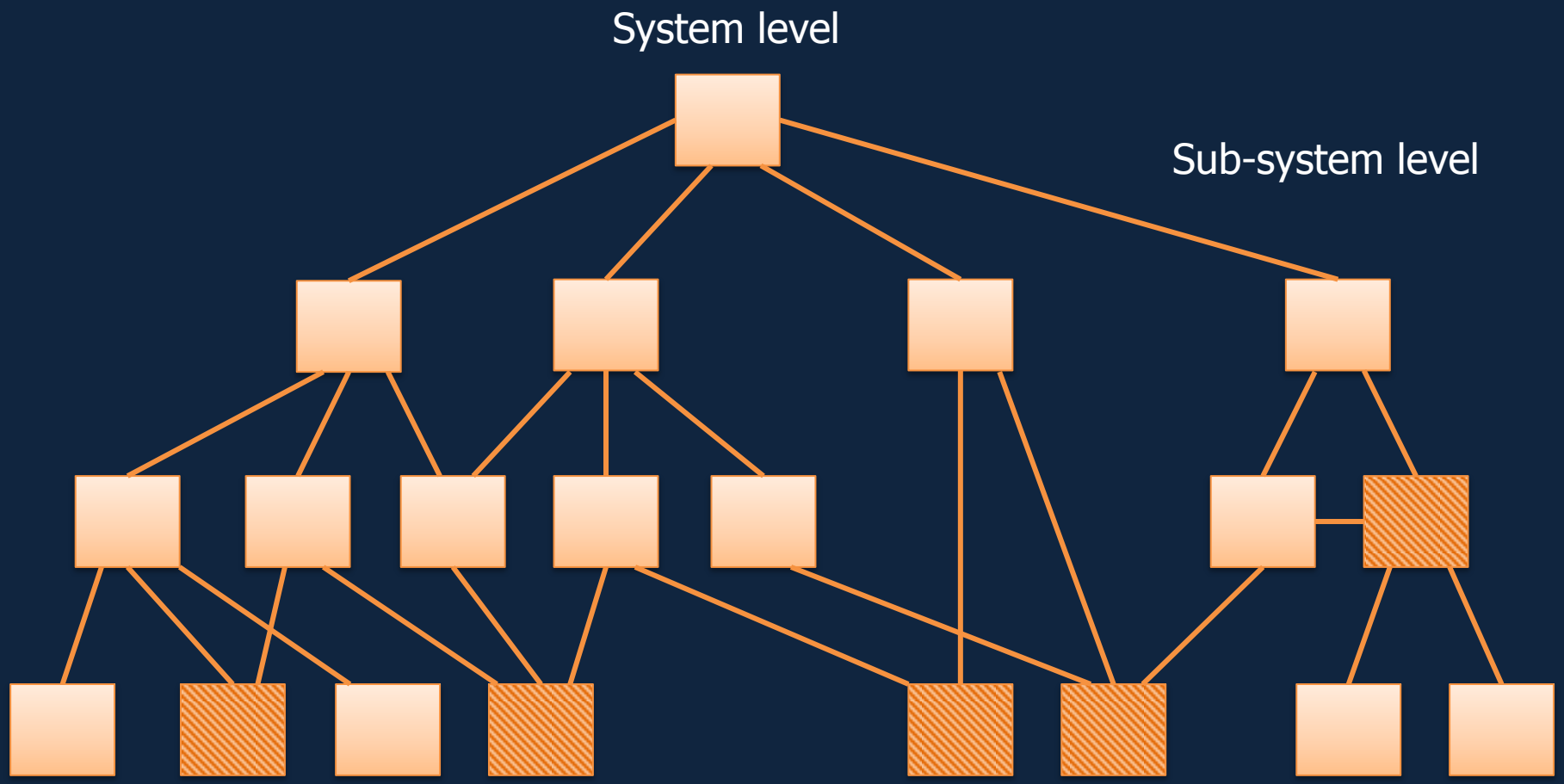
Data structure design – design data structures to hold problem data

Algorithm design – describe algorithms for problem functions

Top-down Design

- In principle, top-down design involves starting at the uppermost components in the hierarchy and working down the hierarchy level by level.
- In practice, large systems design is never truly top-down. Some branches are designed before others. Designers reuse experience (and sometimes components) during the design process.

Hierarchical Design Structure



Design Strategies

- Functional design
  - The system is designed from a functional viewpoint. The system state is centralized and shared between the functions operating on that state.
- Object-oriented design
  - The system is viewed as a collection of interacting objects.
  - The system state is decentralized and each object manages its own state.
  - Objects may be instances of an object class and communicate by exchanging methods.

Mixed-strategy Design

- Although it is sometimes suggested that one approach to design is superior, in practice, an object-oriented and a functional-oriented approach to design are complementary.
- Good software engineers should select the most appropriate approach for whatever sub-system is being designed.

Evolution of Software Design

- Development of modular programs
  - Top-down Refinement
  - Structured Programming
- Translation of data flow or data structure into design definition
- OOP approach to design derivation
- Design patterns for software architecture

Desirable Design Properties

- **Testability** : can the modules/components be easily tested?
- **Maintainability** : is it easy to update the modules in order to bug-fix, extend the features, or adapt the modules to external changes?
- **Diagnosability** : is it easy to trace back occurrence of an error?
- **Reusability** : can the modules be reused in other software projects?
- **Portability** : can the system be easily modified to integrate with new operating environments?
- **Scalability** : is the system capable of dealing with an increased load without requiring much software modifications, and still satisfying the performance requirements?

Important Design Attributes

- **Cohesion** - a measure of how well the contents of a module are related to each other
  - What is a cohesive procedure? a cohesive class?
- **Coupling** – a measure of how interconnected the modules are

- General Goal : High cohesion and low coupling

Cohesion

- A measure of how well a component “fits together”.
- A component should implement a single logical entity or function.
- Cohesion is a desirable design component attribute as when a change has to be made, it is localized in a single cohesive component.
- Various levels of cohesion have been identified.

Cohesion Levels

- Coincidental cohesion (weak)
  - Parts of a component are simply bundled together.
  - A component may have 2 or more unrelated functions.
- Logical association (weak)
  - Components which perform similar functions are grouped.
- Temporal cohesion (weak)
  - Components which are activated at the same time are grouped.

Cohesion Levels

- Communicational cohesion (medium)
  - All the elements of a component operate on the same input or produce the same output.
- Sequential cohesion (medium)
  - The output for one part of a component is the input to another part.
- Functional cohesion (strong)
  - Each part of a component is necessary for the execution of a single function.
- Object cohesion (strong)
  - Each operation provides functionality which allows object attributes to be modified or inspected.

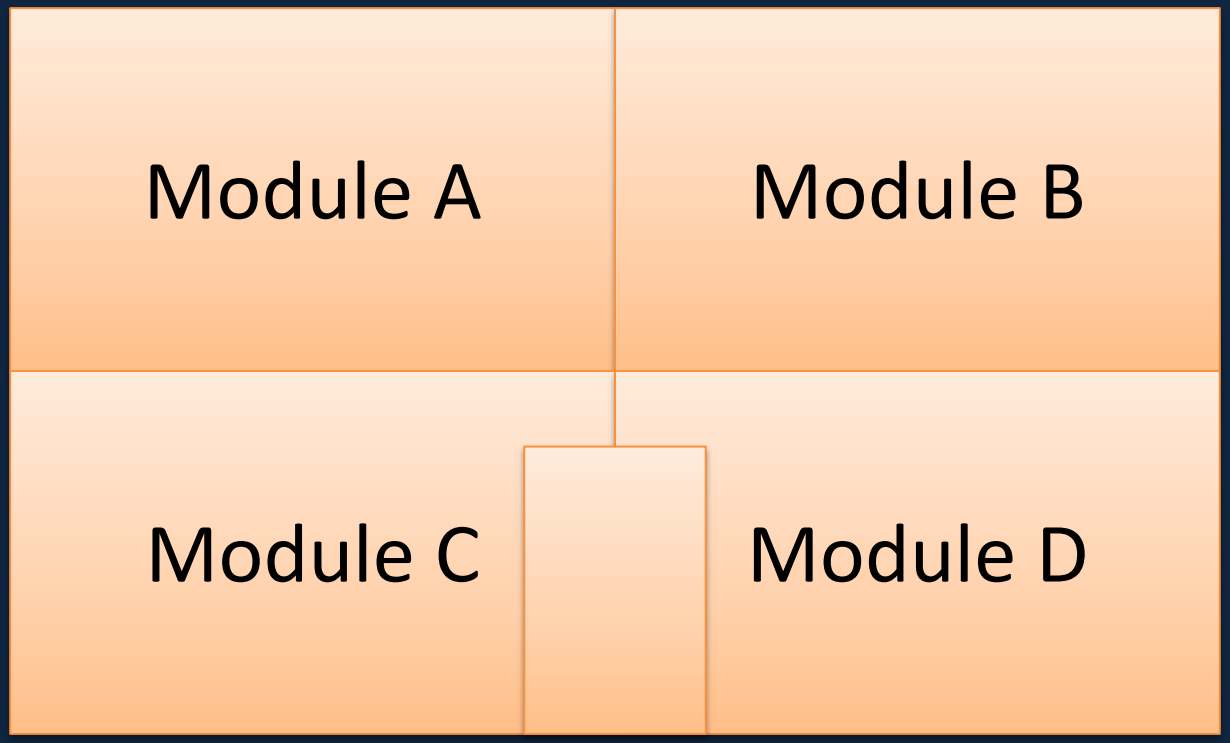
Cohesion as a Design Attribute

- Not well-defined. Often difficult to classify cohesion.
- Inheriting attributes from super-classes weakens cohesion.
  - To understand a component, the super-classes as well as the component class must be examined.
  - Object class browsers assist with this process.

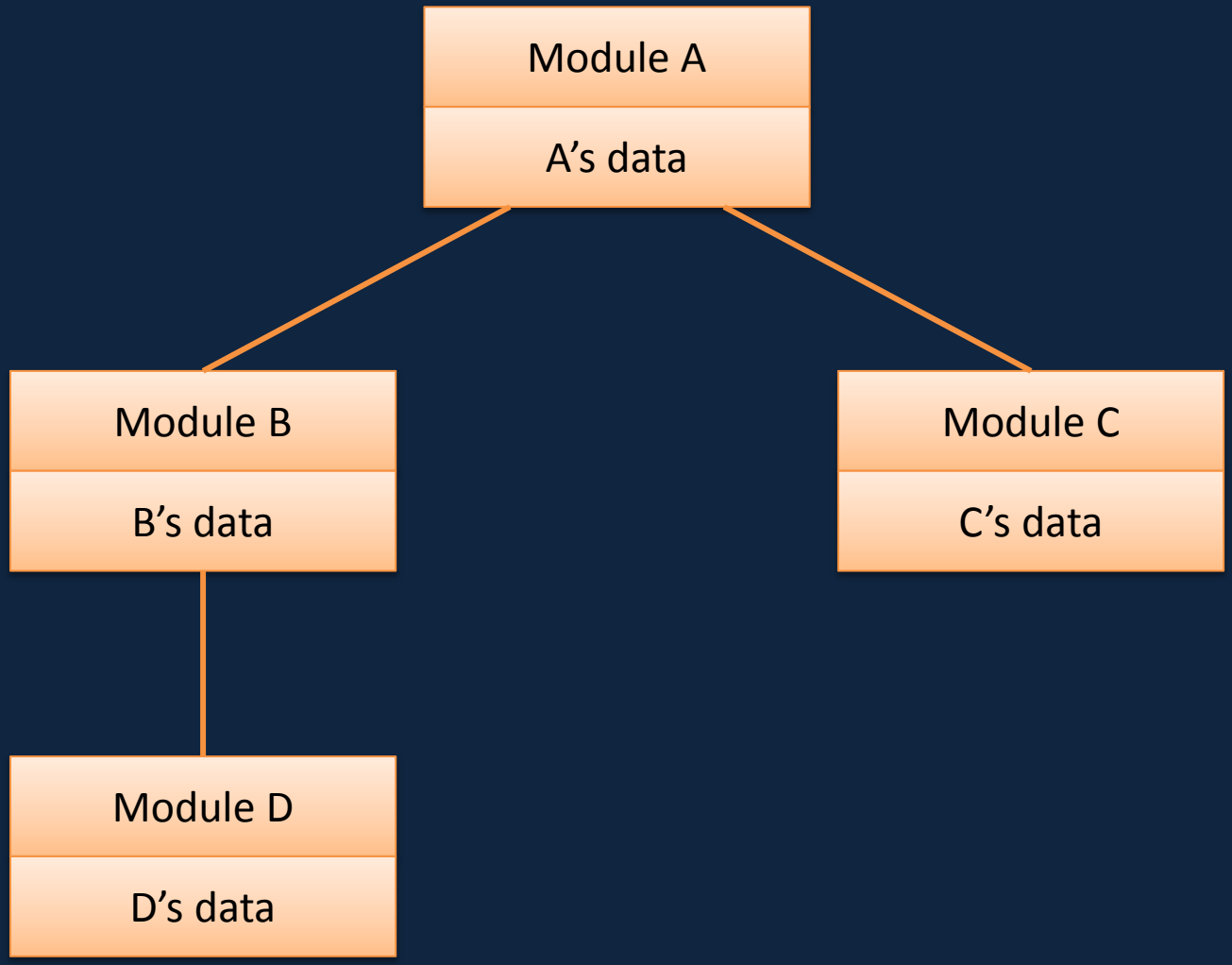
Coupling

- A measure of the strength of the inter-connections between system components.
- Loose coupling means component changes are unlikely to affect other components.
- Shared variables or control information exchange lead to tight coupling.
- Loose coupling can be achieved by state decentralization (as in objects) and component communication via parameters or message passing.

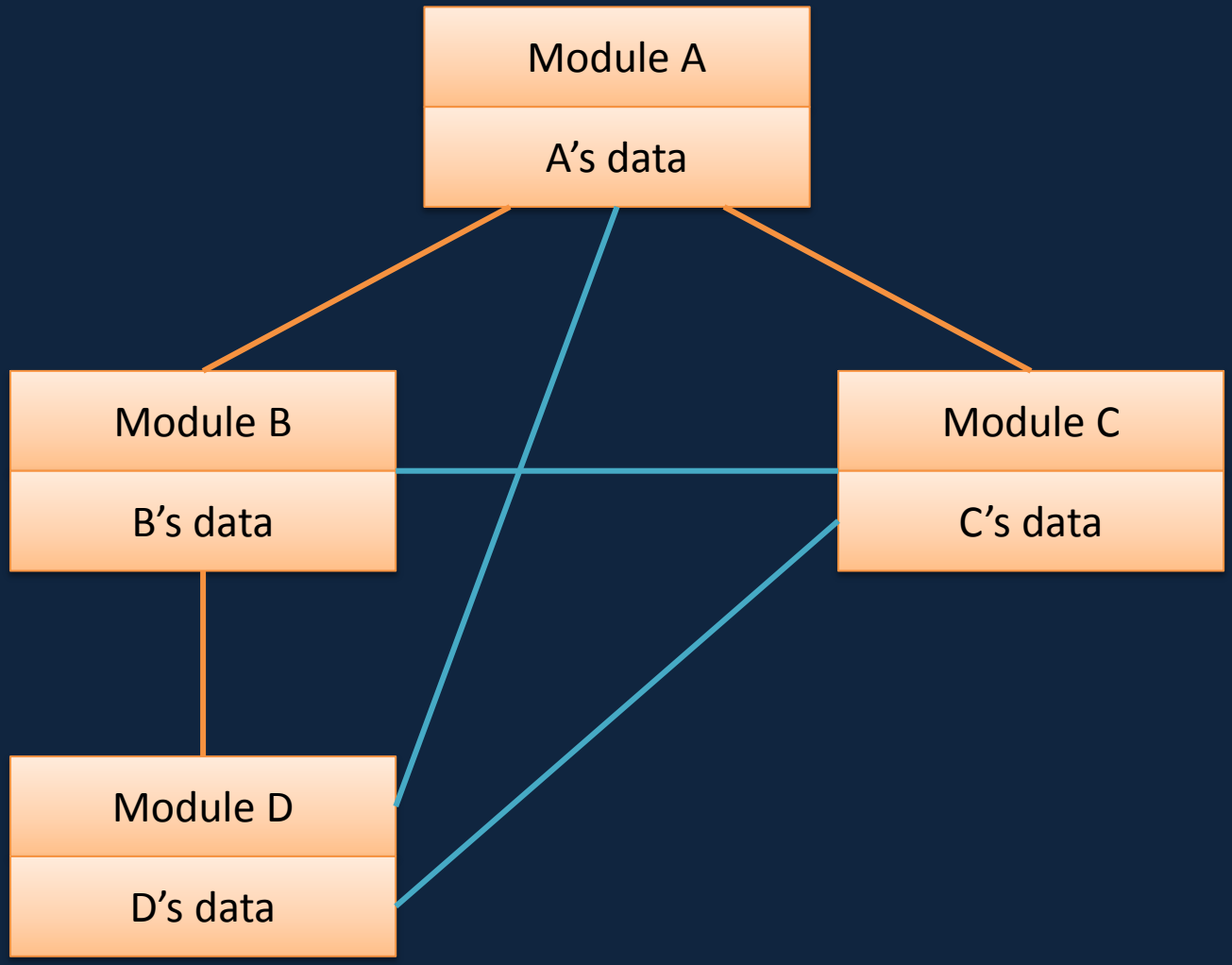
Tight Coupling



Loose Coupling



Tight Coupling



Design and Quality

- 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

Design and Quality

- Some guide questions for assessing the quality of the design
  - 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?
- 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

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

Design Concepts

- 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 low amount of connection needed with other modules

Design Concepts

- 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

References

- I. Sommerville, *Introduction to Software Engineering 9th ed.* Addison-Wesley, 2011
- R. Pressman, *Software Engineering: A Practitioner's Approach*. McGraw-Hill, 2005
- K. Saleh, *Software Engineering.* Cengage Learning Asia, 2010
- D. Gustafson, *Schaum's Outline of Theory and Problems of Software Engineering*, McGraw-Hill, 2002