# Chapter 10

## Component-Level Design

*Slide Set to accompany*

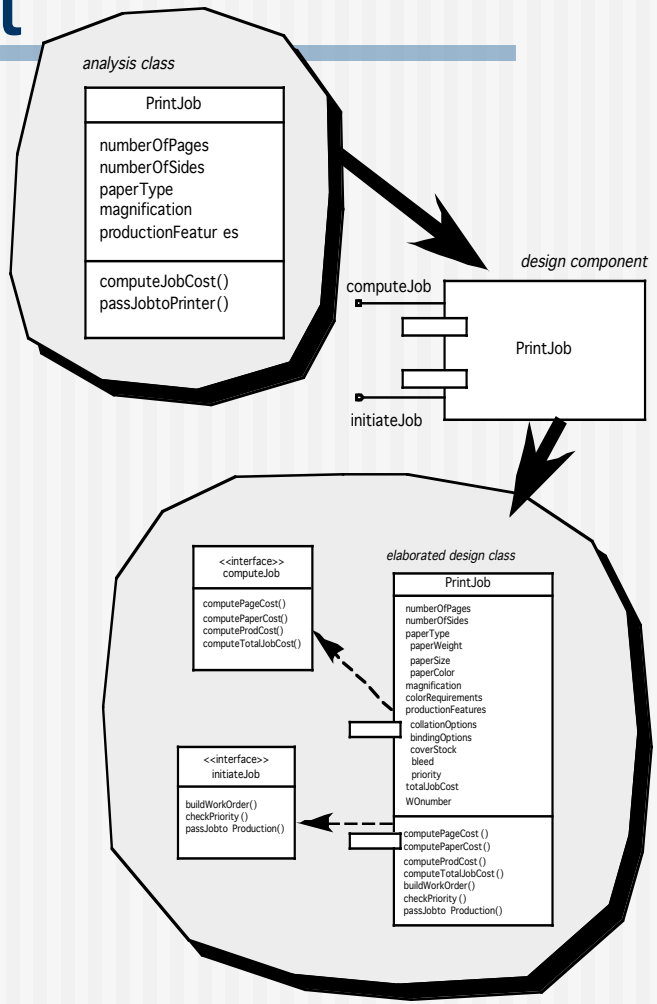
*Software Engineering: A Practitioner's Approach, 7/e*

**by Roger S. Pressman**

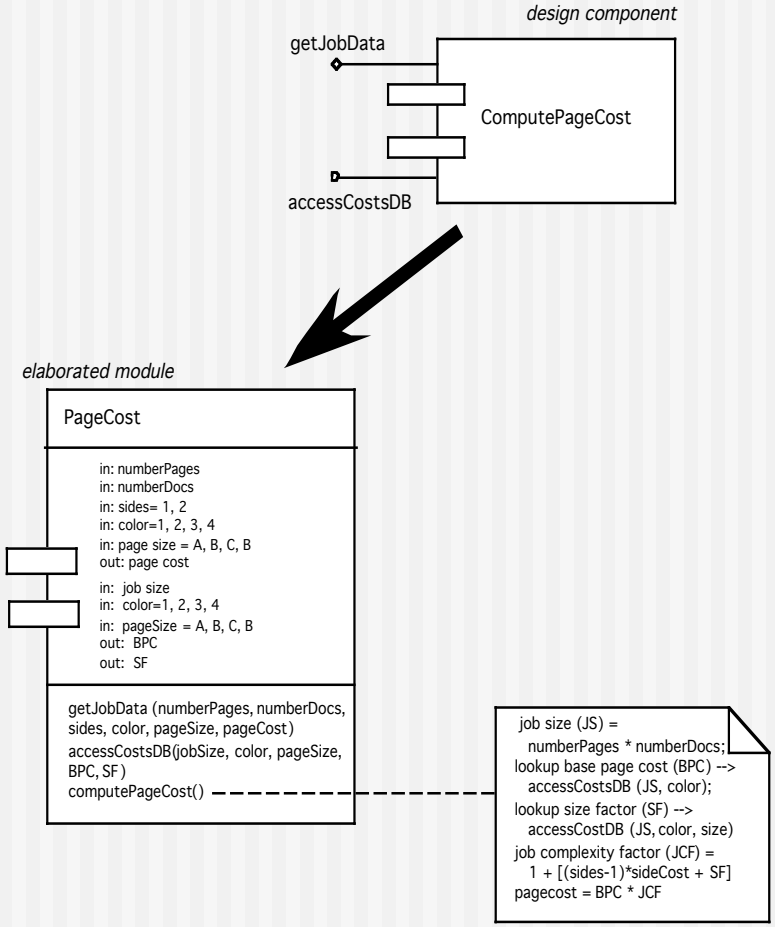
# What is a Component?

- *OMG Unified Modeling Language Specification* [OMG01] defines a component as
  - "… a modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of interfaces.""
- *OO view:* a component contains a set of collaborating classes
- *Conventional view:* a component contains processing logic, the internal data structures that are required to implement the processing logic, and an interface that enables the component to be invoked and data to be passed to it.

# OO Component



# Conventional Component



# Basic Design Principles

- The Open-Closed Principle (OCP). *"A module [component] should be open for extension but closed for modification.*
- The Liskov Substitution Principle (LSP). *"Subclasses should be substitutable for their base classes.*
- Dependency Inversion Principle (DIP). *"Depend on abstractions. Do not depend on concretions."*
- The Interface Segregation Principle (ISP). *"Many client-specific interfaces are better than one general purpose interface.*
- The Release Reuse Equivalency Principle (REP). *"The granule of reuse is the granule of release."*
- The Common Closure Principle (CCP). *"Classes that change together belong together."*
- The Common Reuse Principle (CRP). *"Classes that aren't reused together should not be grouped together."*

Source: Martin, R., "Design Principles and Design Patterns," downloaded from http:www.objectmentor.com, 2000.

# Design Guidelines

- Components
  - Naming conventions should be established for components that are specified as part of the architectural model and then refined and elaborated as part of the component-level model
- Interfaces
  - Interfaces provide important information about communication and collaboration (as well as helping us to achieve the OPC)
- Dependencies and Inheritance
  - it is a good idea to model dependencies from left to right and inheritance from bottom (derived classes) to top (base classes).

# Cohesion

- Conventional view:
  - the “single-mindedness” of a module
- OO view:
  - *cohesion* implies that a component or class encapsulates only attributes and operations that are closely related to one another and to the class or component itself
- Levels of cohesion
  - Functional
  - Layer
  - Communicational
  - Sequential
  - Procedural
  - Temporal
  - utility

# Coupling

- Conventional view:
  - The degree to which a component is connected to other components and to the external world
- OO view:
  - a qualitative measure of the degree to which classes are connected to one another
- Level of coupling
  - Content
  - Common
  - Control
  - Stamp
  - Data
  - Routine call
  - Type use
  - Inclusion or import
  - External

# Component Level Design-I

- Step 1. Identify all design classes that correspond to the problem domain.
- Step 2. Identify all design classes that correspond to the infrastructure domain.
- Step 3. Elaborate all design classes that are not acquired as reusable components.
- Step 3a. Specify message details when classes or component collaborate.
- Step 3b. Identify appropriate interfaces for each component.

# Component-Level Design-II

- Step 3c. Elaborate attributes and define data types and data structures required to implement them.
- Step 3d. Describe processing flow within each operation in detail.
- Step 4. Describe persistent data sources (databases and files) and identify the classes required to manage them.
- Step 5. Develop and elaborate behavioral representations for a class or component.
- Step 6. Elaborate deployment diagrams to provide additional implementation detail.
- Step 7. Factor every component-level design representation and always consider alternatives.