Component Design Principles

Component-Level Design Summary

Component-level design follows architectural design and focuses on defining the internal data structures, algorithms, interface characteristics, and communication mechanisms for system components. It ensures that each component is modular, deployable, and replaceable, encapsulating its implementation while exposing a set of interfaces.

Key Concepts:

- **Component Views**:
  - **OO View**: Components are collaborating classes.
  - **Conventional View**: A component is a functional element with processing logic, internal data structures, and interfaces.
- **Class Elaboration**: This involves detailing attributes, interfaces, and methods for classes before system development begins. Example: "PrintJob" class elaboration includes specifying attributes like weight, size, and color.

Design Principles:

1. **Design by Contract**: Defines the relationship between classes and clients as a formal agreement with rights and obligations.
2. **Open-Closed Principle**: Modules should be open for extension but closed for modification.
3. **Subtype Substitution**: Subclasses should extend but not replace base class functionality.
4. **Dependency Inversion**: High-level modules should depend on abstractions, not low-level modules.
5. **Interface Segregation**: Prefer many specific interfaces over one general-purpose interface.

Cohesion Types:

- **Functional Cohesion**: A module performs one computation and returns a result.
- **Layer Cohesion**: Higher layers access lower layers, but not vice versa.
- **Communicational Cohesion**: Operations that access the same data are grouped within a class.

Coupling Types:

- **Content Coupling**: A component modifies another component’s internal data (violates information hiding).
- **Common Coupling**: Multiple components use a global variable.
- **Routine Call Coupling**: A common occurrence in object-oriented programming.

Component-Level Design Steps:

1. **Identify Design Classes**: From the problem domain and infrastructure classes during architecture.
2. **Class Elaboration**: Specify message details, interfaces, attributes, and processing flow.
3. **Describe Persistent Data Sources**: Identify and describe databases/files that store persistent data.
4. **Elaborate Behavioral Representations**: Model state transitions and actions.
5. **Elaborate Deployment Diagrams**: Represent the location of components.
6. **Refactor and Consider Alternatives**: Redesign and evaluate alternative solutions to improve the model.

Component-level design ensures that software components are well-defined and structured for development, with clear communication and behavioral flow.