

# Component Level Design

- Component? A module or piece of software that serves as a building block
  - Directly or indirectly achieve software objectives

# Component Views

- Object oriented view – set of collaborating classes
  - Data and functions contained in the object
- Conventional view – functional elements, data structures, interface
  - Interface is both the internal and external
- Process view – is the reusable modules and how they are connected
  - Reuse requires management

# Basic Design Principles

- Open-closed principle (OCP) “open to extension, closed to modification”
- Liskov substitution principle (LSP) “subclasses should be substitutable for their base classes”
- Dependency Inversion principle “rely upon abstractions, not concretions”
- Interface segregation principle “many specific interfaces are better than one generic interface”

# Packaging Principles

- Release reuse equivalency principle – what you build for reuse is what you release (you are agreeing to provide a control mechanism)
- Common closure principle – classes that change together should be packaged together
- Common reuse principle – classes that don't rely on one another should not be packaged together

# Component Design

- Use good naming conventions
- Interfaces should be drawn using the simplest notation possible
- Dependencies should be drawn left to right
- Inheritance should be bottom to top

# Cohesion

- Functional – every module does one thing
- Layered – higher level function can access low level functions
- Communicational – all data access/modification is done in a single component
  - These three are necessary and occur quite frequently

# Cohesion cont. (worse)

- Sequential – first component provide input to second, and so on (data is passed)
- Procedural – first calls second, calls third, etc (no data necessarily passed)
- Temporal – state or time based functionality (startup, error, wrap-up, etc)
- Utility – components are grouped together, solely because they are in same category (no other similarities)
  - Statistics

# Coupling

- Coupling is degree of connectedness
  - Higher connectivity = higher complexity
- Content coupling – one component modifies data in another
- Common coupling – using global variables
- Control coupling – module A calls module B, but A “tells” B how to act
- Stamp coupling – some class (B) is a data type in another class (as a parameter)

# coupling

- Data coupling – passing too much data, increases connections = increased complexity
- Routine call coupling – one component calls another component
  - Necessary and frequent
- Type use coupling – component A uses part of component B in its definition