Component-Level Design

Suradet Jitprapaikulsarn

Derived from Roger S. Pressman, *Software Engineering: A Practitioner's Approach*, 6th Edition, McGraw-Hill 2005

What is a Component?

- "A modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of interfaces."

OMG Unified Modeling Language Specification

Three views of components

- Object-Oriented view: a set of collaborating classes
- Conventional view: a functional element of a program
- Process-Related view: reusable unit

Basic Design Principles

- ***The Open-Closed Principle (OCP)***: 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 of abstractions. Do not depend on concretions
- ***The Interface Segregation Principle (ISP)***: Many client-specific interfaces are better than one general purpose interface

Packaging Principles

- ***The Release Reuse Equivalent 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

Component-Level Design Guidelines

- Components
- Interfaces
- Dependencies and inheritance

Cohesion

- Functional
- Layer
- Communicational
- Sequential
- Procedural
- Temporal
- Utility

Coupling

- Content
- Common
- Control
- Stamp
- Data
- Routine call
- Type use
- Inclusion or import
- External

Component-Level Design 1

1. Identify the application domain classes
2. Identify the infrastructure domain classes
3. Refine the design classes
   a. Specify the details of the collaborations
   b. Specify the interfaces of each component
   c. Refine the attributes, data types, and data structures
   d. Describe processing flow

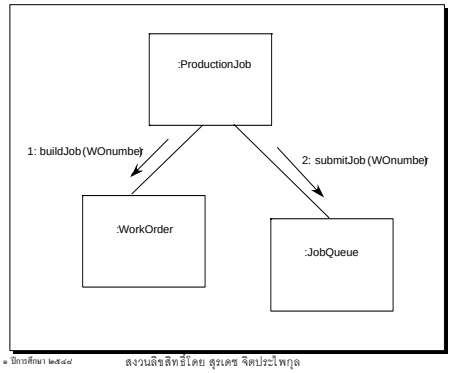
Component-Level Design 2

4. Specify persistent data sources and related classes
5. Refine the behavioral of a class or component
6. Add the implementation detail
7. Refactor

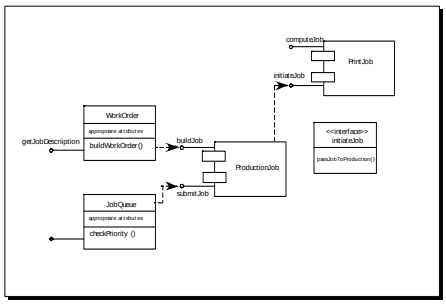
Component Level Design-I

- 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.

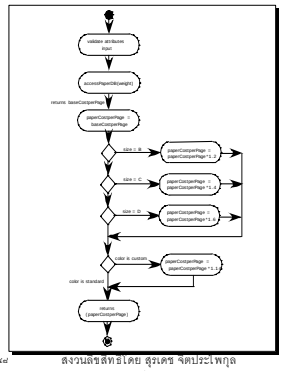
Collaboration Diagram



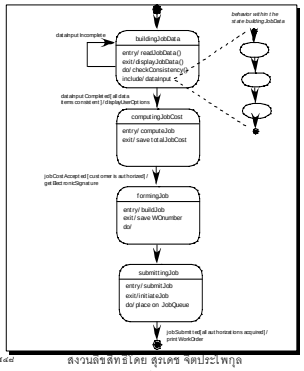
Refactoring



Activity Diagram



Statechart



Object Constraint Language (OCL)

- A formal specification language extension to UML
- A precise text language that provides constraint and object query expressions on an object-oriented model

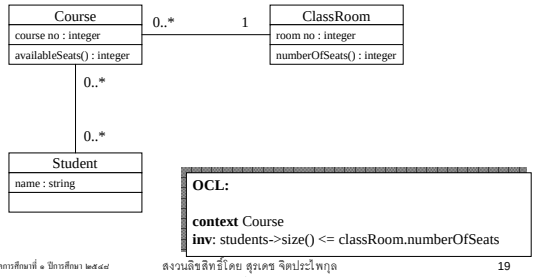
OCL language statements

1. Context
2. Property
3. Operation
4. Keywords

4 Types of constraints

- Invariant
- Pre-condition
- Post-condition
- Guard

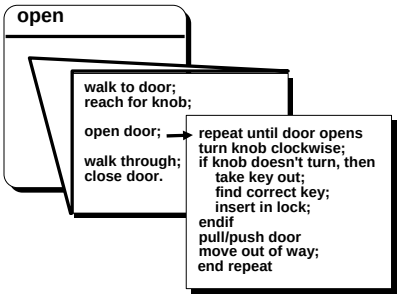
OCL Example



Algorithm Design

- The closest design activity to coding
- The approach
  - review the design description for the component
  - use stepwise refinement to develop algorithm
  - use structured programming to implement procedural logic
  - use 'formal methods' to prove logic

Stepwise Refinement



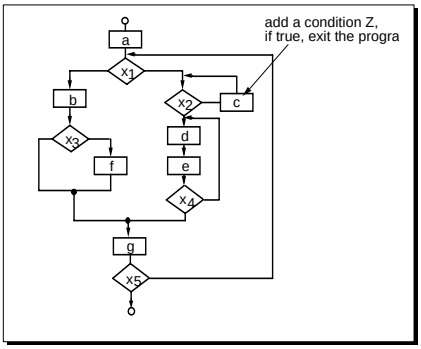
Algorithm Design Model

- represents the algorithm at a level of detail that can be reviewed for quality
- options:
  - graphical (e.g. flowchart, box diagram)
  - pseudocode (e.g., PDL) ... choice of many
  - programming language
  - decision table
  - conduct walkthrough to assess quality

Structured Programming for Procedural Design

- Structured Programming constraints logic flow to
  - Sequence
  - Condition
  - Repetition
- leads to more readable, testable code
- can be used in conjunction with 'proof of correctness
- important for achieving high quality, but not enough

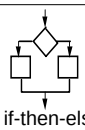
A Structured Procedural D



Decision Table

	Rules					
Conditions	1	2	3	4	5	6
regular customer	T	T				
silver customer			T	T		
gold customer					T	T
special discount	F	T	F	T	F	T
Rules						
no discount	✓					
apply 8 percent discount			✓	✓		
apply 15 percent discount					✓	✓
apply additional x percent discount		✓		✓		✓

Program Design Language (PDL)



if-then-else

```
if condition x
  then process a;
  else process b;
endif
```

PDL

- easy to combine with source code
- machine readable, no need for graphics input
- graphics can be generated from PDL
- enables declaration of data as well as proce
- easier to maintain

Why Design Language

- can be a derivative of the HOL of choi e.g., Ada PDL
- machine readable and processable
- can be embedded with source code, therefore easier to maintain
- can be represented in great detail, if designer and coder are different
- easy to review