

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

เอกสารนี้จัดทำโดย ๑ มีนาคม ๒๕๖๔

สงวนลิขสิทธิ์โดย สุวเดช จิตประไพกุล
ศาล

4

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

เอกสารนี้จัดทำโดย ๑ มีนาคม ๒๕๖๔

สงวนลิขสิทธิ์โดย สุวเดช จิตประไพกุล
ศาล

5

Component-Level Design Guidelines

- Components
- Interfaces
- Dependencies and inheritance

เอกสารนี้จัดทำโดย ๑ มีนาคม ๒๕๖๔

สงวนลิขสิทธิ์โดย สุวเดช จิตประไพกุล
ศาล

6

Cohesion

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

มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าธนบุรี

สงวนลิขสิทธิ์โดย สุรเดช จิกรพรรดิกุล
ศาล

7

Coupling

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

มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าธนบุรี

สงวนลิขสิทธิ์โดย สุรเดช จิกรพรรดิกุล
ศาล

8

Data: If the dependency between the modules is based on the fact that they communicate by passing only data, then the modules are said to be data coupled. In data coupling, the components are independent of each other and communicate through data. Module communications don't contain tramp data. Example-customer billing system.

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

มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าธนบุรี

สงวนลิขสิทธิ์โดย สุรเดช จิกรพรรดิกุล
ศาล

9

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

เอกสารเรียนที่ ๑ ปีการศึกษา ๒๕๖๔

ส่งงานเสร็จสิ้นโดย สุวเดช จิตประไพกุล
ครู

10

Component Level Design-I

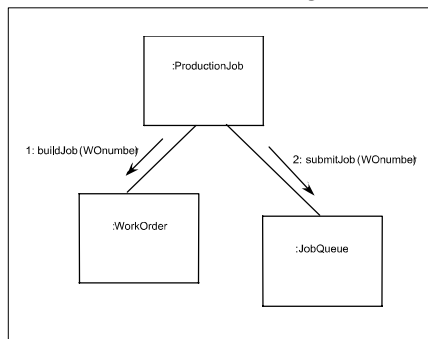
- Step 1. Identify all design classes that correspond to the problem domain.
- Step 2. Identify all design classes that correspond to the infrastructure domain.
- Step 3. Elaborate all design classes that are not acquired as reusable components.
- Step 3a. Specify message details when classes or component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

เอกสารเรียนที่ ๑ ปีการศึกษา ๒๕๖๔

ส่งงานเสร็จสิ้นโดย สุวเดช จิตประไพกุล
ครู

11

Collaboration Diagram



เอกสารเรียนที่ ๑ ปีการศึกษา ๒๕๖๔

ส่งงานเสร็จสิ้นโดย สุวเดช จิตประไพกุล
ครู

12

Refactoring

```

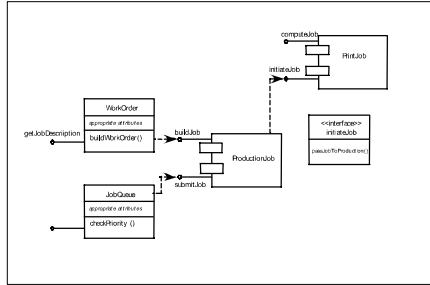
classDiagram
    class ViewOrder {
        getJobDescription()
        getPriority()
        buildOrder()
    }
    class JobQueue {
        getJobDescription()
        checkPriority()
    }
    class Job {
        <<interface>>
        +IProductionJob
    }
    class ProductionJob {
        <<interface>>
        +IJob
    }
    ViewOrder --> Job : getJobDescription()
    ViewOrder --> Job : buildOrder()
    JobQueue --> Job : getJobDescription()
    JobQueue --> Job : checkPriority()
    Job --> ProductionJob : IProductionJob
    ProductionJob --> Job : IJob
    
```

งานออกแบบที่ ๑ ปีการศึกษา ๒๕๔๘

ผลงานลิขสิทธิ์โดย สุนทร จิตประทีปกุล

หน้า ๑

13



ภาคการศึกษาที่ ๑ ปีการศึกษา ๒๕๔๘

สงวนลิขสิทธิ์โดย สุรเดช จิตรประไพกุล
กาล

13

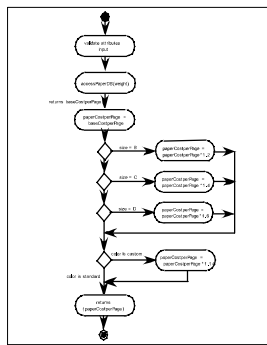
Activity Diagram

```

graph TD
    Start(( )) --> AddNewCustomer[add new customer]
    AddNewCustomer --> AddNewCustomer1[add new customer (add new customer)]
    AddNewCustomer1 --> AddNewCustomer2[add new customer (add new customer)]
    AddNewCustomer2 --> Decision{ }
    Decision --> AddNewCustomer3[add new customer (add new customer)]
    Decision --> AddNewCustomer4[add new customer (add new customer)]
    Decision --> AddNewCustomer5[add new customer (add new customer)]
    AddNewCustomer3 --> Join(( ))
    AddNewCustomer4 --> Join
    AddNewCustomer5 --> Join
    Join --> AddNewCustomer6[add new customer]
    AddNewCustomer6 --> AddNewCustomer7[add new customer (add new customer)]
    AddNewCustomer7 --> End(( ))
  
```

Diagram illustrating an Activity Diagram (UML) for a process flow. The diagram shows a sequence of activities and decision points:

- Start** (Initial node)
- add new customer** (Activity)
- add new customer (add new customer)** (Activity)
- add new customer (add new customer)** (Activity)
- Decision** (Diamond shape) with three outgoing paths:
 - add new customer (add new customer)** (Activity)
 - add new customer (add new customer)** (Activity)
 - add new customer (add new customer)** (Activity)
- Join** (Diamond shape) where the three paths merge.
- add new customer** (Activity)
- add new customer (add new customer)** (Activity)
- End** (Final node)



ภาคการศึกษาที่ ๑ ปีการศึกษา ๒๕๔๘

สงวนลิขสิทธิ์โดย สุรเดช จิตประไพกุล
ศาล

14

Statechart

```

stateDiagram-v2
    [*] --> buildingInfoData
    buildingInfoData --> computeAgendaCost : add new job to agenda
    computeAgendaCost --> FormingJob : add new job to agenda
    FormingJob --> submitJobInfo : add new job to agenda
    submitJobInfo --> [*] : add new job to agenda
    buildingInfoData --> [*] : cancel job from agenda
    computeAgendaCost --> [*] : cancel job from agenda
    FormingJob --> [*] : cancel job from agenda
    submitJobInfo --> [*] : cancel job from agenda
  
```

Statechart diagram illustrating a process flow:

- buildingInfoData** (initial state):
 - entry: read jobData()
 - exit: write jobData()
 - do: checkConsistentAgenda()
 - do: add new job to agenda
- computeAgendaCost**:
 - entry: computeAgendaCost()
 - exit: save to JobDataCost
- FormingJob**:
 - entry: buildJob()
 - exit: save JobNumber
 - do:
- submitJobInfo**:
 - entry: submitJobInfo()
 - exit: save JobDataCost
 - do: add new job to agenda

Transitions and Events:

- From **buildingInfoData** to **computeAgendaCost**: add new job to agenda
- From **computeAgendaCost** to **FormingJob**: add new job to agenda
- From **FormingJob** to **submitJobInfo**: add new job to agenda
- From **submitJobInfo** to **buildingInfoData**: add new job to agenda
- From **buildingInfoData** to **computeAgendaCost**: cancel job from agenda
- From **computeAgendaCost** to **FormingJob**: cancel job from agenda
- From **FormingJob** to **submitJobInfo**: cancel job from agenda
- From **submitJobInfo** to **buildingInfoData**: cancel job from agenda

Labels at the bottom of the diagram:

สถานะเริ่มต้น = เริ่มต้นระบบ
 สถานะสุดท้าย = จบระบบ

Diagram illustrating a Statechart (UML State Machine Diagram) for a system, likely related to job scheduling or resource allocation. The diagram shows four states and their transitions:

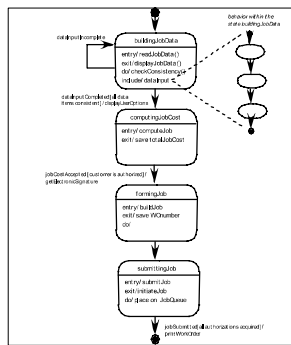
- buildingInfoData** (Initial State):
 - entry: read jobData()
 - exit: write jobData()
 - do: checkConsistentAgenda()
 - do: add new job to agenda
- computeAgendaCost**:
 - entry: computeAgendaCost()
 - exit: save to JobDataCost
- FormingJob**:
 - entry: buildJob()
 - exit: save JobNumber
 - do:
- submitJobInfo**:
 - entry: submitJobInfo()
 - exit: save JobDataCost
 - do: add new job to agenda

Transitions and Events:

- From **buildingInfoData** to **computeAgendaCost**: add new job to agenda
- From **computeAgendaCost** to **FormingJob**: add new job to agenda
- From **FormingJob** to **submitJobInfo**: add new job to agenda
- From **submitJobInfo** to **buildingInfoData**: add new job to agenda
- From **buildingInfoData** to **computeAgendaCost**: cancel job from agenda
- From **computeAgendaCost** to **FormingJob**: cancel job from agenda
- From **FormingJob** to **submitJobInfo**: cancel job from agenda
- From **submitJobInfo** to **buildingInfoData**: cancel job from agenda

Labels at the bottom of the diagram:

สถานะเริ่มต้น = เริ่มต้นระบบ
 สถานะสุดท้าย = จบระบบ



ภาคการศึกษาที่ ๑ ปีการศึกษา ๒๕๔๘

สงวนลิขสิทธิ์โดย สุรเดช จิตประไพกุล
ศาล

15

[illegible]