Lecture 5: Design Principles

Review so far…

- W1: Intro ✓
- W2: Modelling ✓
- W3-4: Design patterns ✓
- **W5-6: Design principles & system architecture**
- W7-8: Testing
- W9: Continuous integration
- W10: Review

Today's Plan

- 10:05-10:55:
  - Quiz 5 (assessed, correctness)
  - Main prin ion
  - PI 5 (asses
- Break
- 11:05-11:55:
  - …PI 5 (assessed, participation)
  - Other principles: Abstraction, flexibility, testability …
- Long break
- 13:00-14:00 OR 14:00-15:00 in BO1028: Lab 5 Design Principles

Quiz 5

- Join YACRS:
  - Session: 1
- It's design principle quiz ti

Q5.1 Coupling and Cohesion

A software engineer should always aim to:

- A: Increase both coupling and cohesion
- B: Decrease both coupling and cohesion
- C: Increase coupling and decrease cohesion
- D: Decrease coupling and increase cohesion

Q5.1 Coupling and Cohesion (solution)

A software engineering should always aim to:

- A: Increase both coupling and cohesion
- B: Decrease both coupling and cohesion
- C: Increase coupling and decrease cohesion
- **D: Decrease coupling and increase cohesion [correct]**

Q5.2 Abstraction

A software engineer should aim to:

- A: Keep the level of abstraction as high as possible
- B: Keep the level of abstraction as low as possible
- C: Keep a balanced level of abstraction
- D: Not worry about abstraction

Q5.2 Abstraction

A software engineer should aim to:

- **A: Keep the level of abs high as possible – maintains flexibility an e options during development [correct]**

- B: Keep the level of abstraction as low as possible
- C: Keep a balanced level of abstraction
- D: Not worry about abstraction

Q5.3 Packages

Java packages are:

- A: Not used
- B: Difficult to represent in UML
- C: A key way to represent the divide and conquer principle
- D: Only able to have a low level of cohesion

Q5.3 Packages

Java packages are:

- A: Not used
- B: Difficult to represent in UML
- **C: A key way to represent the divide and conquer principle [correct]**
- D: only able to have a low level of cohesion

Design

Properties of "good" design:

- Increasin … …creasing revenue
- Ensuring t … …the requirements
- Accelerating development

- Increasing qualities such as
  - Usability
  - Efficiency
  - Reliability
  - Maintainability
  - Reusability
  - …

Why are design principles important?

- Design principles provides a framework for talking about design deci (rational?) decisions (on a slightl (e.g. patterns)

- You are probably already us these, giving these principles the proper names will improve your usage of them

- They can provide rationale for design decisions to make sure you come up with a "good" design
- They will improve your working practice and make your programming, managing, teaching life easier

DESIGN Principles

- **Divide and Conquer**
- **Cohesion**
- **Coupling**
- **Keep the level of abstraction as high as possible (alre**
- Increase reus
- Reuse existing designs and c
- Anticipate obsolescence
- Design for portability
- Design for testability
- Design defensively

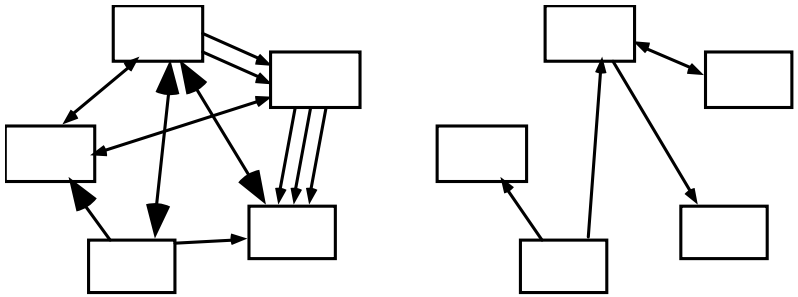
Divide and conquer

Divide and conquer (combine)

- Trying to deal with something big all at once is normally much hard ies of smaller things art.
  - Separate
  - An individual software eng cialize.
  - Each individual componen nd therefore easier to understand.
  - Parts can be replaced or changed without having to replace or extensively change other parts.

Coupling and Cohesion

- Cohesion
  - How rela
- Coupling
  - How dependent classes ar ner (between classes)
- Design to **increase cohesion** and **decrease coupling**



Cohesion

High Cohesion is Good

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

Functional Cohesion

- Keep related functions together and keep everything else out.

- No side-effects (same resul n the same input at any point, no interaction)

- Benefits to the system:
  - Easier to understand
  - More reusable
  - Easier to replace

Layer Cohesion

- Lower level layers provide services to higher level layers via a well
  - Separate e packages

Communicational Cohesion

- Grouping together classes and methods for accessing the same
  - For exam
    ling to logs, a specific database

Sequential, Procedural, and Temporal Cohesion

- All relate to functionalities that work together, but all refer to slightly different relationships.

- **Sequential** – Order of sequence is important because one method a
  - For example, input from prompt for following methods

- **Procedural** – Methods are calle another but don't depend on each other
  - For example, functions with prepare lines of a log file.
- **Temporal** – Methods must be called together at a specific time
  - For example, a startup routine.

Utility Cohesion

- Grouping modules that are re-used but don't have a natural ho
  - For exam

Coupling review

Coupling: High Coupling is Bad

- Content
- Common
- Control
- Data
- Routine
- Type Use
- Import
- External

Content Coupling

- Content from one module to another is not masked, but accessed

- Values from one object ar irectly by another or cached

  - Make your instance values private
  - Use getters/setters to control access

Content Coupling - example

Imagine a student class with a student ID attribute.

```
class Student(String )
{
        public
        …
}

Student myStudent = new Stud       ”8”);

// Option A:
myStudent.studentId = “0000000000000000”;
>> [nothing]

// Option B:
myStudent.setStudentId(“0000000000000000”);
>> “Invalid ID, too long”
```

Common Coupling

- Global attributes shared between components
  - Compon e become coupled
    - Difficu
    - Low Readability
    - Changes to the global t many modules

Control Coupling

- The behaviour of a module is controlled by flags
  - Issue: On s and logic of another module

- Solution: Polymorphism,…

Control Coupling

```
…
public drawRoutine(Command command) {
        if (command.equals("Draw Circle") {
                drawCircle();
        }
        else {
                drawRectangle(
        }
} …
```


Refactored using polymorphishm (introducing stamp coupling)…

```
public drawRoutine(Shape shape) {
        shape.draw()
}
```

Control Coupling

```
…
public runCode(String name) {
        if (IGNORE_CODE) {return;}
        …
} …
```



```
…
public myCode(String name) {
        if (DEBUG_CODE) {
                System.out.print(name);
        }
        …
} …
```

Stamp Coupling

- Stamp coupling occurs when a data type is used as a parameter to an method.
  - If a module takes an object argument and only uses a very small amount of the

For example:

```
produceReport(Student student)
```

```
produceReport(String studentID)
```

Stamp Coupling


```
public interface Identifier
{
  public abstra
  public abstra
}

public class Student implements                    …}
public class Employee implements Identifier {…}

public class Report
{
  public void produceReport(Identifier o)
  {...}
  ...
}
```

Data coupling

- Data coupling occurs when many variables are given as simple arguments – t parse all of them.
- The more arg ish en this coupling (and comple
- Stamp (object) vs Data (simple rade-off

```
…
enrollStudent(String studentName, int studentId,
        Enrollment enrollmentRole)

enrollStudent(Student student)
…
```

Routine Coupling

- Routine coupling occurs when several functions need to be called t

- Possible solution: reduce ro upling by writing a single routine that encapsulates the sequence.

Routine Coupling - Example

```
…
myTouchPoint.drawBorder(x,y);
myTouchPoint.drawBackground(x,y);
myTouchPoint.drawShadow(x,y);
…
```

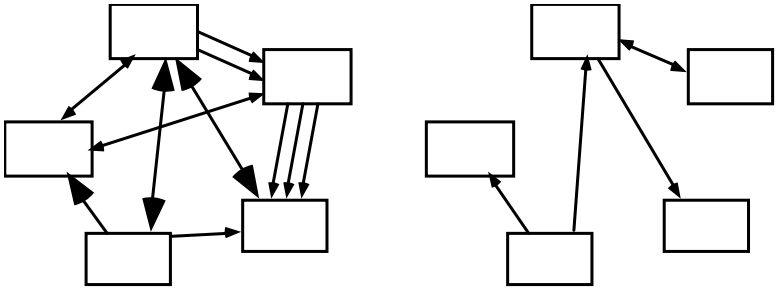

OR

```
myTouchPoint.draw(27,56);
…
public void draw(Int x, Int y) {
        this.drawBorder(x,y);
        this.drawBackground(x,y);
        this.drawShadow(x,y);
}
```

Type Use Coupling

- If a class uses a data type defined in another class, this introduces

- Can you implement an ass thout a type coupling ? Perhaps use in the way and a pattern…



Import and External Coupling

- import coupling occurs when one module imports the functionalit
- external coupling occurs w lication depends on the configuration of the t.
  - Sometimes the Façade pattern can help

Break

PI 5

- YACRS session: 1217
- On which line does content occur?

PI 5.1 Coupling and cohesion exam question

PI 5.1 Exam question

- **Routine Coupling**, Lines must be called together to create intended effect, Lines 6 -7, 15-16
  - Solution: write a separate function to handle this, e.g. easy to change file format etc
- **Common Coupling**, LOG VERBOSITY LEVEL, IGNORE_USER_REQUEST could be a global variable, Lines 6-7, 15-16
  - Solution:
- **Control Couplin** les) appear to be passed in to control this funct d need to know the logic of the flags
  - Normally, mem object could e.g. clean-up
  - Line 4: Move the logic outside
- **Data Coupling**: Many attributes or provided when arguments could be simplified, Lines 9 to 13
  - Solution: Parse in object, at least only parse necessary arguments
- **Content Coupling**: Attributes are accessed directly rather than through a defined function or interface, Line 9-13
  - Solution: parse in object and ensure to get and set on private.
- **External**: Memory structure seems to be assumed

Summary

- Design principles (especially cohesion and coupling)
- General pr
  - Modelling techniques from visualize and describe a domain and it
  - The design patterns you have seen in W3-4 can help minimize coupling – but not necessarily required to ensure low coupling

- Lab 5 at 13:00 or 14:00 in BO1028

Next Week

- Design Principles (cont.) and **System Architecture (including**

- Read:
  - Chapter 9.5 >