Lecture 09: Fundamental Principles and Best Practices for Software Design

Valentin Razmov

Outline

- Best practices for software design (cont. from Lecture 08)
- Principles for good design
- Software systems building good practices
- Examples of bad design practices
- How detailed should a design be?

Resources

- *Code Complete* (*2nd ed.*), chapter 5, by Steve McConnell, http://www.cc2e.com/docs/Chapter5-Design.pdf
- *Design Patterns Explained – A New Perspective on Object-Oriented Design*, by Alan Shalloway and James Trott
- *Agile Software Development – Principles, Patterns and Practices*, by Robert C. Martin

How to Approach Design (a reminder)

"Treat design as a wicked, sloppy, heuristic process. Don't settle for the first design that occurs to you. Collaborate. Strive for simplicity. Prototype when you need to. Iterate, iterate, and iterate again. You'll be happy with your designs."

-- Steve McConnell, *Code Complete* (*2nd ed.*), ch.5

"There are two ways of constructing a software design: one way is to make it so simple that there are obviously no deficiencies; the other is to make it so complicated that there are no obvious deficiencies."

-- C.A.R. Hoare (1985)

Best Practices for Software Design (cont.)

- Favor composition over inheritance.
  - Example:

Principles for Good Design: Single Responsibility Principle

- "A class should have only one reason to change."
- Principle of strong cohesion
- "God-object" metaphor
- Example 1: Putting state in a GUI class.
  - Model-View-Controller pattern helps to avoid this.
- Example 2: Where is this principle violated below?

```
interface Modem {
  public void dial (String pno);
  public void hangup();
  public void send (char c);
  public char recv();
}
```

Principles for Good Design: Open-Closed Principle

- "Software entities (classes, modules, functions, etc.) should be open for extension but closed for modification."
- Example: An abstract class to extend (with as many new subclasses as needed) rather than modifying an existing class to accommodate each new addition.
- The designer chooses what changes to anticipate and what parts of the system to "fix."

Principles for Good Design: Interface Segregation Principle

- "Clients should not be forced to depend on methods that they do not use."
- Example: Dogs jump but don't sing. J

Principles for Good Design: Dependency Inversion Principle

- (A) "High-level modules should not depend on low-level modules. Both should depend on abstractions."
- (B) "Abstractions should not depend on details. Details should depend on abstractions."
- Example: Separation of policy and mechanism

Principles for Good Design: Dependency Inversion Principle (Example)

```
#define TEMP_GAUGE 0x86
#define FURNACE 0x87
#define ENGAGE 1
#define DISENGAGE 0

void Regulate(
  double minTemp, double maxTemp)
{
  for(;;) {
    while (read(TEMP_GAUGE)>minTemp)
      wait(1);
    set(FURNACE, ENGAGE);
    while (read(TEMP_GAUGE)<maxTemp)
      wait(1);
    set(FURNACE, ENGAGE);
  }
}
```

```
void Regulate(
    Thermometer t, Heater h,
    double minTemp,
    double maxTemp)
{
  for(;;) {
    while (t.Read() > minTemp)
      wait(1);
    h.Engage();
    while (t.Read() < maxTemp)
      wait(1);
    h.Disengage();
  }
}
```


Principles for Good Design: Liskov Substitution Principle

- "Subtypes must be substitutable for their base types."
- This is different from saying that there must be an IS-A relationship between the two types.
  - Example: Is *Square* always substitutable for *Rectangle*?

Examples of Bad Designs Practices

- "Design by committee"
  - Everyone on the committee puts in their favorite features into the soup. What is the result?
  - Moral: The design must be owned and managed by someone.
- Other examples you have heard of?

Additional Best Practices for Building Software Systems

- Make the common case fast and the uncommon case correct.
  - But do not spend time on optimizing code early on.
  - "*It is easier to optimize correct code than to correct optimized code.*"-- Donald Knuth
- Establish and maintain a clear audit trail.
  - It requires little investment upfront but is invaluable for debugging purposes.

How Detailed Should It Be?

Table 5-2 Design Formality and Level of Detail Needed

Factor	Level of Detail Needed in Design Before Construction	Documentation Formality
Design/construction team has deep experience in applications area.	Low Detail	Low Formality
Design/construction team has deep experience but is inexperienced in the applications area.	Medium Detail	Medium Formality
Design/construction team is inexperienced.	Medium to High Detail	Low-Medium Formality
Design/construction team has moderate-to-high turnover.	Medium Detail	—
Application is safety-critical.	High Detail	High Formality
Application is mission-critical.	Medium Detail	Medium-High Formality
Project is small.	Low Detail	Low Formality
Project is large.	Medium Detail	Medium Formality
Software is expected to have a short lifetime (weeks or months).	Low Detail	Low Formality
Software is expected to have a long lifetime (months or years).	Medium Detail	Medium Formality

Your Questions on Principles and Practices for Software Design