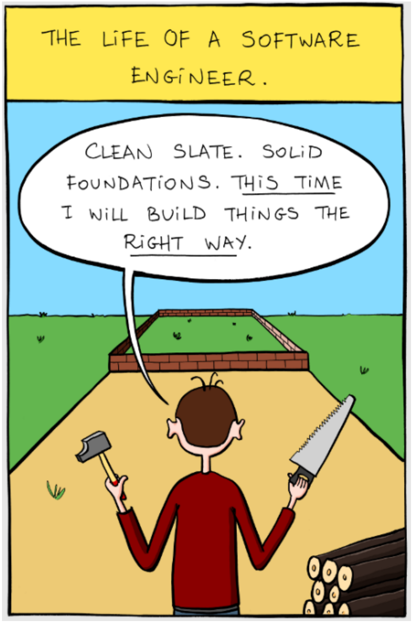
AMSS Lecture 3: Design Patterns (I)

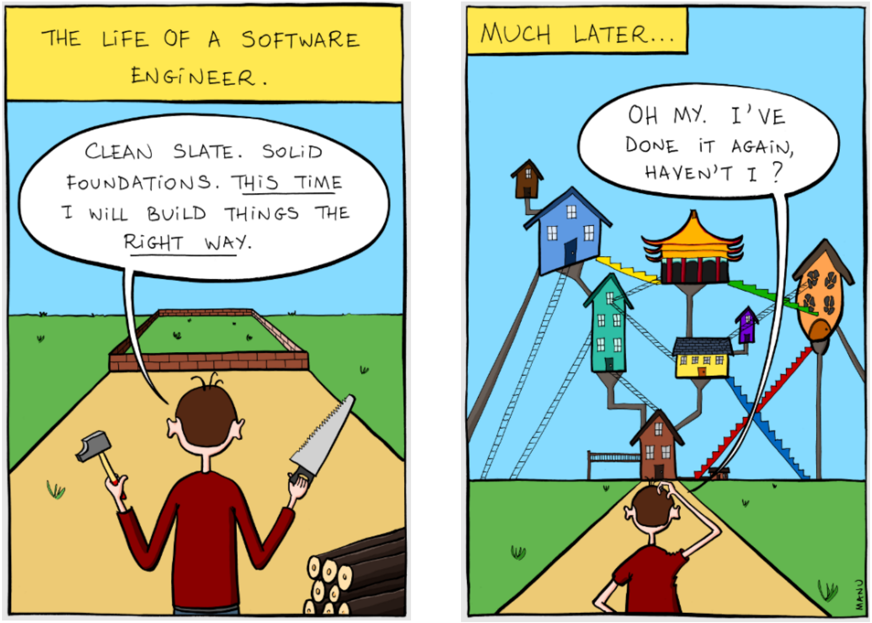
Virgil-Nicolae Șerbănuță

2025

Agenda

1. What are Design Patterns?
2. Classification of Patterns
3. **Iterator Pattern**
4. **Builder Pattern**
5. **Singleton Pattern**
6. Wrap-up


THE LIFE OF A SOFTWARE ENGINEER.
CLEAN SLATE. SOLID FOUNDATIONS. THIS TIME I WILL BUILD THINGS THE RIGHT WAY.


THE LIFE OF A SOFTWARE ENGINEER.
CLEAN SLATE. SOLID FOUNDATIONS. THIS TIME I WILL BUILD THINGS THE RIGHT WAY.
MUCH LATER...
OH MY. I'VE DONE IT AGAIN, HAVEN'T I ?
MANU

What Are Design Patterns?

Definition

Reusable solutions to common software design problems.

Origin

Popularized by the “Gang of Four” (Gamma, Helm, Johnson, Vlissides, 1994).

Purpose

- Provide shared vocabulary
- Improve code maintainability
- Promote reusability and clarity

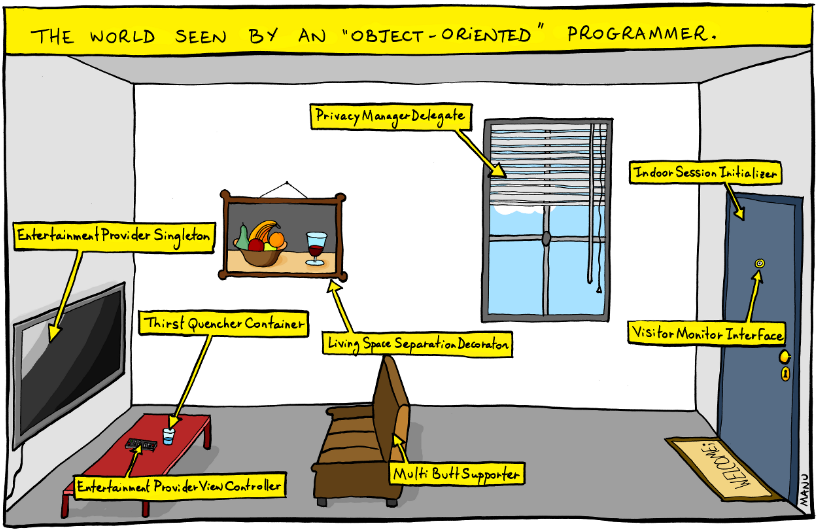
Example

Instead of reinventing how to traverse a collection, we apply the **Iterator** pattern.

Pattern Classification

Design patterns are typically grouped into three main categories:

Category	Description	Example Patterns
Creational	How objects are created	Singleton, Builder, Factory Method
Structural	How classes and objects are composed	Adapter, Bridge, Decorator
Behavioral	How objects interact and communicate	Iterator, Observer, State


THE WORLD SEEN BY AN "OBJECT-ORIENTED" PROGRAMMER.
Privacy Manager Delegate
Indoor Session Initializer
Entertainment Provider Singleton
Thirst Quencher Container
Living Space Separation Decorator
Visitor Monitor Interface
Entertainment Provider View Controller
Multi Butt Supporter
WELCOME!
MANU

Iterator Pattern

Type
Behavioral pattern

Intent
Provide a way to access elements of a collection sequentially without exposing its internal structure.

Problem Solved
How to traverse a collection (e.g., list, tree, array) without knowing its implementation?

Solution
Define an `Iterator` interface with methods like `hasNext()` and `next()`.

Iterator Pattern code example

Source file

```java
// Step 1: Create the Iterator interface
interface Iterator {
    boolean hasNext();
    Object next();
}

// Step 2: Create the Container interface
interface Container {
    Iterator getIterator();
}

// Step 3: Create a concrete class implementing Container
class NameRepository implements Container {
    private String[] names = {"Alice", "Bob", "Charlie", "

    @Override
    public Iterator getIterator() {
```