Overview of Reuse Techniques

- Design Patterns
- Architecture Patterns
- Frameworks
- API and Libraries
- Encapsulation of Legacy Systems
- COTS
- Generative Programming
- Component-based Development
- Feature-Oriented Programming
- Model-driven Development
- Service-Oriented Architecture
- Configurable Applications
- Software Product Lines
- Aspect-Oriented Software Development
- Model-driven Development
Library, API, and Framework
Library (API)

- Application Programming Interface (API)
- Libraries (or API) implement services used by several kinds of programs
  - It is a common way of software reuse
- API makes available common functions
  - To convert data from common formats (e.g., string to integer)
  - To access resources, files, DB, etc.
  - Abstract data types, such as queue, stack
import java.util.Vector;

public class Customer {

    String name;
    Vector phoneNumbers = new Vector();

    void removePhoneNumber(String c){
        phoneNumbers.removeElement(c);
    }

    void addPhoneNumber(String c){
        phoneNumbers.addElement(c);
    }

    ...
}
Framework

- Framework is a general structure
- Framework is an incomplete system
  - It is composed of classes and interfaces that implement the system standard structure
- A system is implemented by adding classes to fill in the gaps (i.e., hot spots)
  - For instance, abstract class in a framework should be extended in a system
Main Properties of Framework

- **Inversion of control**
  - Unlike in libraries or applications, the program flow of control is dictated by the framework

- **Extensibility**
  - A framework can be extended by the user

- **Non-modifiable framework code**
  - The framework code should not be modified

- **A framework has a default behavior**
Classification of Frameworks

- Infrastructure Frameworks
  - Support the creation of infrastructure for systems, such as compiling environment

- Integration Frameworks
  - Support exchanges of messages among components (communication)

- Application Frameworks
  - Support development of new applications with common characteristics, such as Web applications
Framework Extension

- Frameworks are large entities which are supposed to be extended

- Examples of framework extension
  - Include new concrete classes that extend abstract ones
  - Override methods that implement standard behavior
  - Include configuration files, such as XML
Main Drawback

- Framework is usually large and complex
  - It is hard to understand
  - It may take long time to be effectively used
  - Developers may need only a simple functionality
Design Patterns
Design Patterns

- A design pattern is a general reusable solution to a common problem.
- Patterns are known best practices:
  - They allow reuse of knowledge from experts.
- They do not describe a complete solution, since it is supposed to be reused in different applications.
Elements of a Design Pattern

- **Name**
  - It is an identifies

- **Problem Description**

- **Solution**
  - It is a *template* of the solution and can be used in different applications

- **Consequences**
  - Results you get when you apply the pattern
Example of Problem
The Observer Design Pattern

- **Name**
  - Observer

- **Problem Description**
  - It separates the subject of its representations

- **Solution (next slide)**

- **Consequences**
  - It optimizes updates from and notification to the viewers
Observer Solution

```
Subject
- Attach (Observer)
- Detach (Observer)
- Notify ()

Observer
- Update ()

ConcreteSubject
- GetState ()
- subjectState

ConcreteObserver
- return subjectState
- Update ()
- observerState

for all o in observers
  o -> Update ()

observerState = subject -> GetState ()
```
MDD and Code Generation
Motivation

- Reuse of code is usually hard to achieve
  - It involves loads of details which are language and platform dependents
MDD Solution

- Model-driven Development (MDD)

- It aims to raise the abstraction level
  - Reuse of models instead of code

- By means of code generators, the system code is automatically created
Why Reuse of Models?

- Models are expected to last longer
- Models facilitate communication among developers
  - They are sometimes used to communicate with customers
- Models are usually developed in mature software process
  - With or without code generation
The MDD Approach

- It proposes the development, reuse, maintenance and evolution at design phase

- Reuse of models is still in maturing stage (not very frequent in industry)
  - It is more common in specific domains or in research center or academia
MDD Process

- Models are software independent
  - As, high-level code is hardware independent
Models can be compiled to a high-level programming language
- Models can be (partially) reused in different contexts
MDD Steps

1. Select some of existing models
2. Choose parts of the models you want
   - It might be necessary to adapt a model
   - It might be necessary to create new models
3. Integrate all partial models of a system
4. Choose a implementation technology
   - It might be necessary to describe the mapping from models to implementation
5. Generate the system code
Current Drawbacks

- MDD is still immature
  - For instance, it lacks support from tools and development environments
- Models are usually seen as superfluous
  - Code is the main asset
- Developers resist to MDD
  - They prefer programming than modelling
  - They are afraid of losing their jobs as programmers
Bibliography

  - Chap. 16 Software Reuse
  - Section 5.5 Model Driven Engineering