

Reuse

- One of the primary reasons that people are interested in Object Technology is that it enables reuse.
- Reuse can lead to faster, easier development (shorter time to market).
- Reuse promotes higher quality.
- Code reuse in OO comes in two forms:
 - Delegating.
 - Inheriting.

1

What Artifacts Are Reusable?

- Short answer: anything.
 - Architectures.
 - Analysis models.
 - Design models.
 - Code.
 - Test plans.
 - Test cases.
 - Documentation.

2

Kinds of Reuse

- Parts (constructing from prefabricated building blocks).
 - Procedures.
 - Classes.
 - Components.
- Frameworks (customizing existing software artifacts).
 - Inversion of control.
- Patterns (reusing plans).

3

Real World Analogy: Building Houses

- Parts – Pre-fabricated or modular construction.
- Frameworks – Remodeling an existing house.
- Patterns – Building multiple houses in a development with the same blueprints.

4

Reusing Parts (1 of 2)

- Creation of the parts.
 - Mining.
 - Fabrication.
- Finding reusable parts.
 - Libraries, searching.
- Certification of parts.
 - Testing.
 - Hardening.
- Building with the parts.
 - Glue code.

5

Reusing Parts (2 of 2)

- Levels of visibility:
 - Black box - you can't see inside.
 - Clear box - you can see inside and make changes.
 - Translucent box - you can see inside but you can't make changes.

6

Component Reuse

- Each component realizes an interface specification.
- To use the component, you must invoke its operations according to the interface.
- For maximum benefit, the components will be large, and the glue code that invokes them will be small.

7

Frameworks

- A *framework* is a partially completed application.
- The missing pieces must be filled in.
- The framework specifies the interfaces for the missing pieces.

8

Framework Reuse

- Just the opposite of component reuse.
 - The part that is reused is the glue.
 - The part you have to write is the components.
- For maximum benefit, the framework (glue) would be large, and the customization would be small.

9

Developing Frameworks (1 of 2)

- Frameworks are developed to simplify construction of software in particular application domains.
 - For example, there are frameworks for banking, insurance, manufacturing, graphics, video games, e-commerce, etc.
- Frameworks:
 - Frequently are outgrowths of successful application development efforts.
 - May be built from scratch (e.g., Eclipse).

10

Developing Frameworks (2 of 2)

- Most of the detailed classes that are specific to the original application are removed.
- Only the generally reusable classes are left in the framework.
- Properties may be removed from classes to make them less specific.
- Properties may be added to classes to make them more general.

11

Patterns

- A pattern is an abstraction of a recurring structure. (A common solution to a recurring problem).
- Patterns are discovered, not invented.
 - We learn from encountering similar problems and find that there are common effective solutions.
 - We distill out the essence of these solutions (how they are similar), and filter out their differences.

12

Where We Find Patterns

- Patterns help in all sorts of problem solving.
 - Tailoring - patterns for cutting cloth and sewing the pieces together.
 - Architecture - patterns for arranging rooms in a building, buildings in a city.
 - Music - patterns for constructing musical compositions (e.g., chord progressions, construction of symphonies, marches, jazz)
 - Writing - patterns for poetry, plays, business letters, etc.

13

Patterns In Software

- Architectural patterns.
- Analysis patterns.
- Design patterns.
- Programming patterns.

14

Some Programming Patterns

- Counter.
- Accumulator.
- Iterator.
- Guard.

15

Architecture Patterns – Layered Architectures

- Three tier
 - Presentation.
 - Model.
 - Persistence.
- Two tier thick client.
 - Presentation & Model in client.
 - Persistence in server.
- Two tier thin client.
 - Presentation in client.
 - Model and persistence in server.

16

More Architecture Patterns

- Pipes and filters.
 - Dataflow.
 - Tightly coupled.
- Message queuing.
 - Middleware.
 - Loose coupling.

17

Analysis Patterns (1 of 2)

- Specification classes (a.k.a. type object).
 - The specification class factors out properties that are common to subsets of instances of another class.
- Composite pattern.
 - Hierarchical structure of object instances.
- State pattern.
 - Object has different properties (attributes, operations, links, constraints) in different states.

18

Analysis Patterns (2 of 2)

- Façade.
 - Single point of contact.
 - Simplifies usage.
- Proxy.
 - Stand-in for something else.
 - Allows for change.
 - Proxy can manage the communication (caching, logging, etc.)

19

Design Patterns

- Three categories:
 - Creational - creating objects or data.
 - Prototype.
 - Singleton.
 - Structural - maintaining relationships.
 - Adapter.
 - Composite.
 - Façade.
 - Proxy.
 - Flyweight.
 - Behavioral - managing collaborations.
 - Iterator.
 - Strategy.
 - Command.
 - Interpreter.
 - Mediator.
 - Observer.
 - State.

20

References

- Gamma, Helm, Vlissides, and Johnson, *Design Patterns, Elements of Reusable Object-Oriented Software*, Reading, MA: Addison-Wesley, 1995.
- Fowler, *Analysis Patterns*, Reading, MA: Addison-Wesley, 1997.
- Cooper, *Java Design Patterns*, Reading, MA: Addison-Wesley, 2000.
- Shaw and Garlan, *Software Architecture*, Upper Saddle River, NJ: Prentice-Hall, 1996.
- http://www.objenv.com/cetus/oo_patterns.html