CS220

Software Maintenance

Definition

- Software maintenance
  - The process of changing a software system after it has been delivered

Reasons behind maintenance

- Why delivered software needs change?

Reasons behind maintenance

- Why delivered software needs change?
  - Fault repair
  - Platform adaptation
  - System enhancement (adding functionality features)

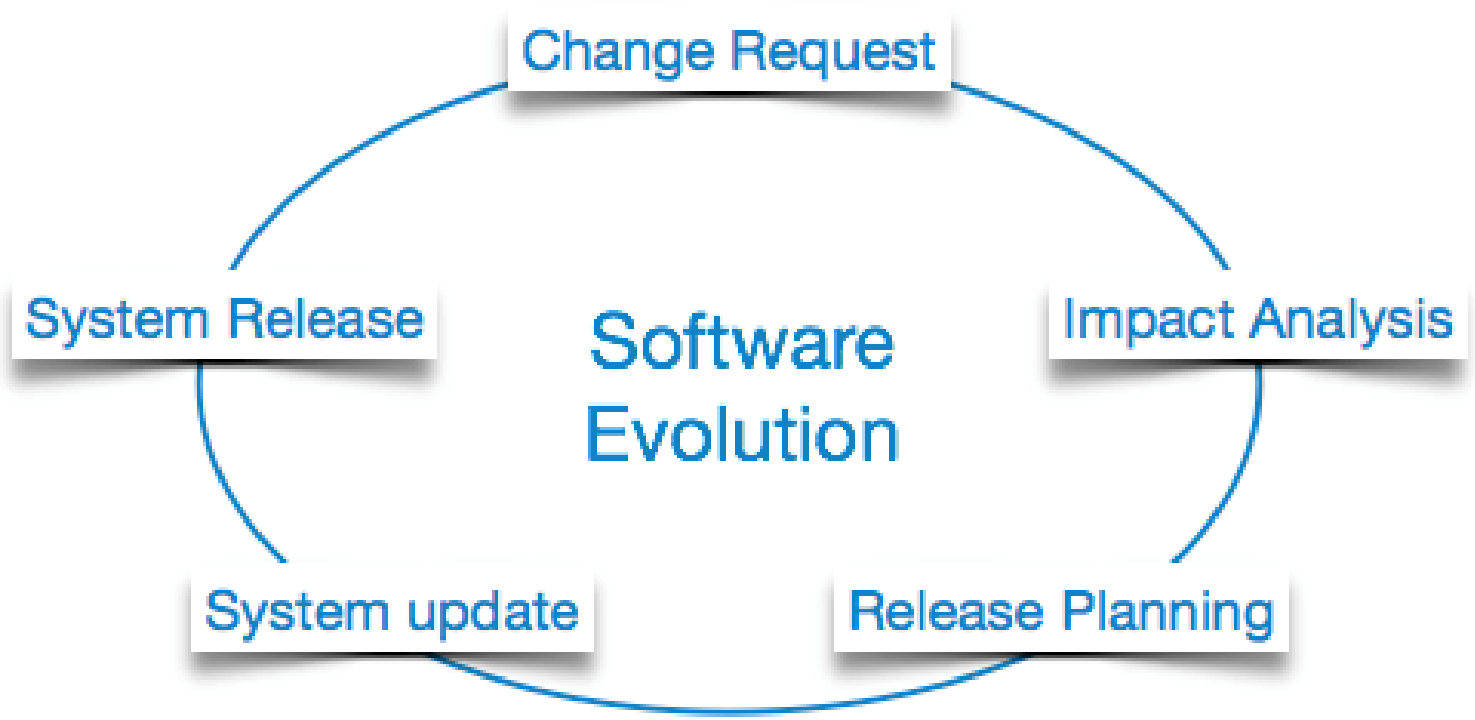
Maintenance is important

- 60%--80% of overall IT cost
  - Software is too expensive to discard after one version

A big picture

- Initial development
- Software evolution
- Software servicing
- Phase-out phase

A smaller picture --- evolution process



Software reengineering

- Redocumenting
- Structure/architecture refactoring
- Programming language translation
- Data reengineering

When to stop supporting a software

When to stop supporting a software

- Business value
- Maintenance expense

Design patterns

Design Patterns

Elements of Reusable Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides

Foreword by Grady Booch

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

What are design patterns

- Solutions to specific problems in OO software design
- 23 patterns in 3 categories
  - Creational
  - Structural
    - Composite
    - …
  - Behaviorial
    - Observer
    - Interpreter
    - …

Why are we studying them?

Observer

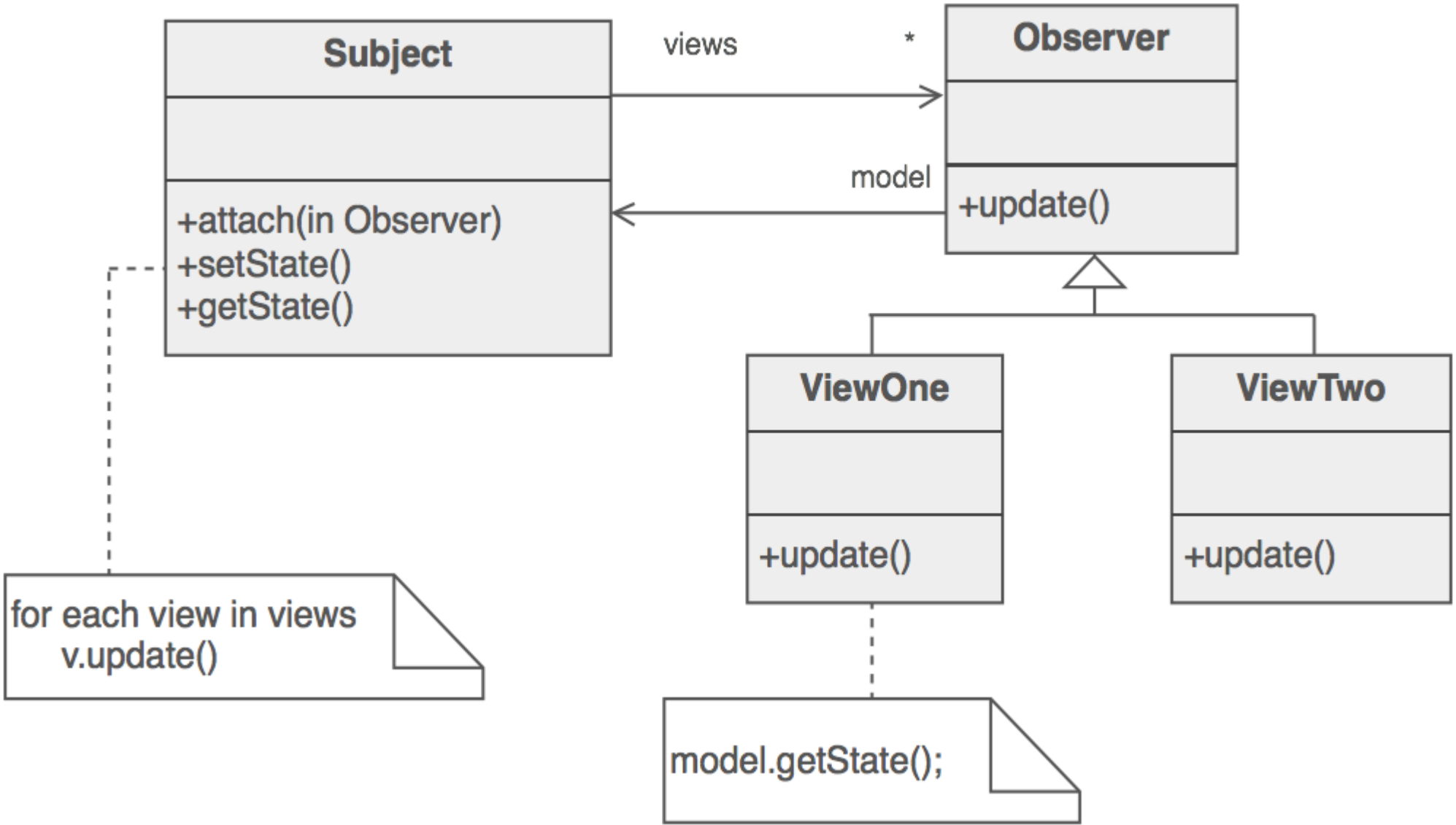
- One to many relationship
  - The many need to know changes in “one” immediately

- Example
  - Points & Shapes
  - Map & location-based services
  - A game character & other game components
  - …

Example

- If a person/subject changes its status, how to let all his “subscriber” knows?
  - What to do when there is only one subscriber?
  - What to do when there are multiple subscribers of different types?
  - What if new subscribers are added?
  - How to make the code easy to maintain and extend?

Class diagram



Example (location, location-related service)

- “location” would be the *Subject* in previous slide
- “*observer”* would be the superclass of all the sub-classes that try to update themselves based on the location information

The benefit of observer pattern

- When new types of observers are added, the prototype and implementation of the subject class doesn’t need any changes.

Other things to pay attention

- Don’t forget the subscribing and unsubscribing methods
- Pull notification vs push notification
- What if I want to delete a subject
- Can an observer subscribe multiple subjects?

Composite pattern

- Tree hierarchy
- How do you build a tree?

How to build a tree and traverse it?

```
struct node{
   struct node* left;
   struct node* right;
   int val;
   int sum(){
        …
   }
}
```

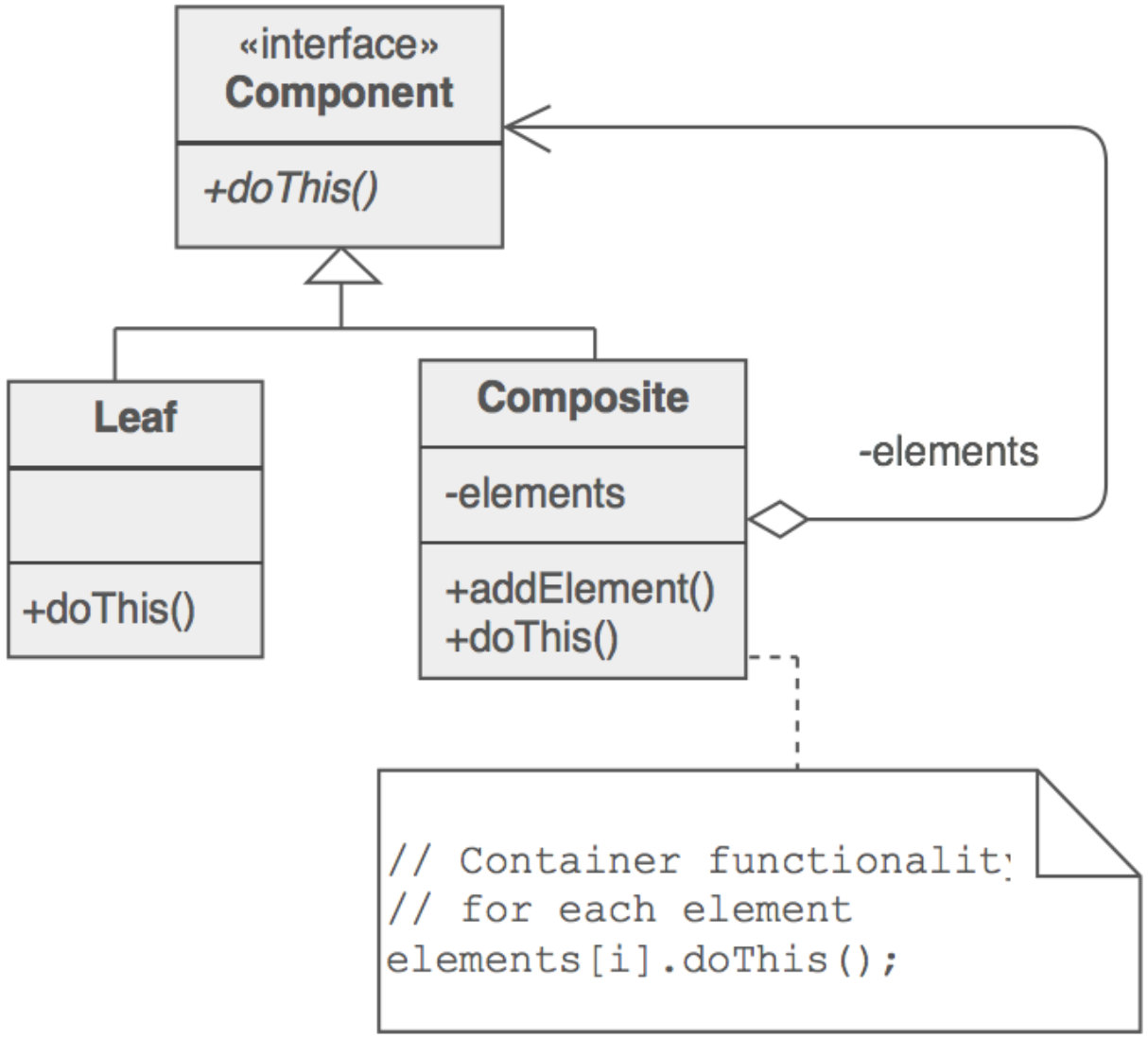
How to differentiate leaves and others?

```
struct leaf{
   int val;
   int sum(){ return val;}
}
```

How to accommodate different types of internal nodes?

- Examples
  - struct node or struct leaf?
  - Book
  - Graphics

Class diagram



Apply composite pattern to tree

- “Leaf” in previous slide is tree leaf
- “Composite” in previous slide is non-leaf nodes in a tree

Interpreter

- What is an interpreter
  - Language, compiler

- Example
  - Boolean expression
    - Abstract syntax tree

    a && b || !c

    a parser will turn this into an abstract syntax tree, and then an interpreter will evaluate the tree. How to write a program to do the tree-based evaluation?

How to do addition & subtraction

- How to represent an addition expression?
  - Constant + Constant
  - Constant + Constant + Constant
- How to represent a subtraction expression?

How to do addition & subtraction

- How to represent an addition expression?
  - Tree is a good form
- How to represent a subtraction expression?
  - Tree

- The challenge:
  - Any node in the above tree could be a constant, an addition expression, or a subtraction expression, etc.

Class diagram