OOAD – UNIT II

Design Patterns

- GRASP Principles
- Design and use case realization
- Design patterns

Responsibility-Driven Design (RDD)

Detailed object design is usually done from the point of view of the metaphor of:

- Objects have responsibilities
- Objects collaborate

Responsibilities are an abstraction.

The responsibility for persistence.

Large-grained responsibility.

The responsibility for the sales tax calculation.

More fine-grained responsibility.

The 9 GRASP Principles

- Creator
- Information Expert
- Controller
- Low Coupling
- High Cohesion
- Polymorphism
- Pure Fabrication
- Indirection
- Protected Variations

Information Expert

"expresses the common ?intuition? that objects do things related to the information they have.'

Assign the responsibility for ?knowing?[something] to the object (class) that has the information necessary to fulfill the responsibility.

Creator

Assign to class B the responsibility to create an instance of class A if

- B aggregates A objects.
- B contains A objects.
- B records instances of A objects.
- B closely uses A objects.
- B has the initializing data that will be passed to A when it is created

GRASP: Polymorphism
Principle

When related alternatives or behaviors vary be type (class), assign responsibility for the behavior?using polymorphic operations?to the types for which the behavior varies.

PATTERN: Controller

Problem: What object in the domain (or application coordination layer) receives requests for work from the UI layer?

System operations (see SSD): major input events to the system

The controller is the first object beyond the UI layer that is responsible for receiving or handling a system operations message.

Note that UI objects delegate system operation request to a controller.

PATTERN: Controller

Solution: Assign the responsibility to a class representing one of the following choices:

A facade controller, which represents

- the overall system
- A root object
- A device that the software is running within, or
- A major subsystem

A use case or session controller which represents a use case scenario in which the system operation

Occurs

Encapsulation

A programming/design languagemechanism.

A packaging / scoping mechanism for names

Names can refer to data, types, ..

Especially, a means of packaging data.

Information Hiding

Design principle by which a module is assigned a “secret”.

A module’s secret is usually

A design decision.

What type of design decisions might we want to hide from the clients of a module?

Cohesion

Measure of the degree of ?relatedness? that elements within a module share.

Degree to which the tasks performed by a single module are functionally related.

Brain storm:

Why put procedures/methods

together within a module/class?

Coupling

Measures the degree of dependency that exists between modules.

A uses a service/method m of B

A passes on an object o returned from B.m()

A provides visibility of B to C by returning a reference to B in A.getB()

A.m(B b, ?)

A calls C.m(B b ?) which expects a B object

A class X in A has an attribute of type Y defined in B

A.m() changes an attribute in B via B.setX()

A.m() changes a (public) attribute in B directly via assignment

A changes a ?flag? in B (ie an attribute which controls execution decisions in B; ie

behaviour of B as seen by others)

A and B both refer to an object o, and A can change o

Pure Fabrication

Problem: Existing objects, ie domain objects, are not appropriate to have the responsibility

Solution suggested by Information Expert not appropriate

Might violate high cohesion, low coupling

Solution: Fabricate (ie create, make up) a new object to hold the responsibility

GRASP: Pure Fabrication

Assign a highly cohesive set of responsibilities to an artificial or convenience class that does not represent a problem domain concept?somethingmade up, to support high cohesion, low coupling, and reuse.

Can you think of an example from EA?

Design of objects

Representational decomposition

Behavorial decomposition

It?s OK for OO software to have objects representing behaviour, ie use case, process, function, strategy,

TS But do not overdo it

All GoF design patterns are Pure Fabrications

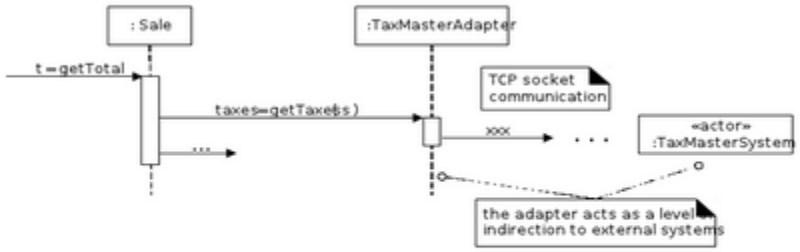
Indirection

Problem: How to assign responsibility to avoid direct coupling? How to de-couple objects?

Solution: Assign responsibility to an intermediatory object to mediate between the two components

Example: Adapter pattern

Intermediatory to external tax calculators



Indirection

“Most problems in computer science can be solved by another level of indirection”

GRASP Protected Variations

Problem:

How to design objects, subsystems, and systems so that the variations or instability in these elements does not have an undesireable impact on other elements?

Solution:

Identify points of predicted variation or instability; assign responsibility to create a stable interface around them.

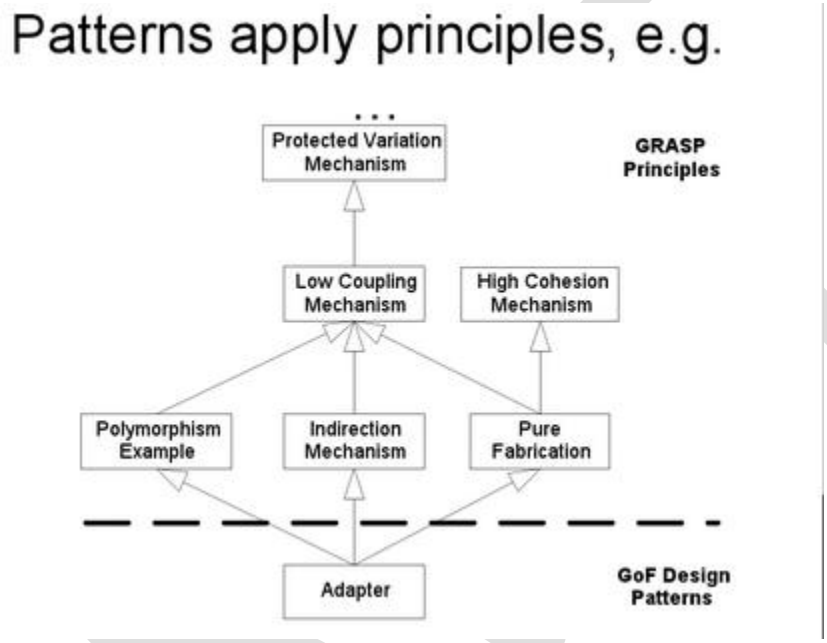
Core PV Mechanisms

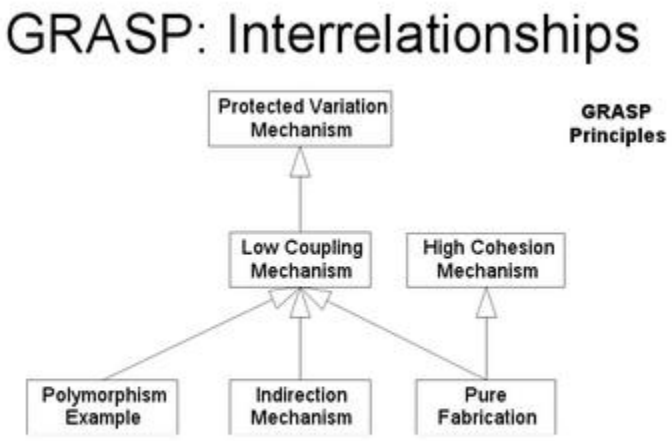
Encapsulation.

Interfaces.

Polymorphism.

Indirection,

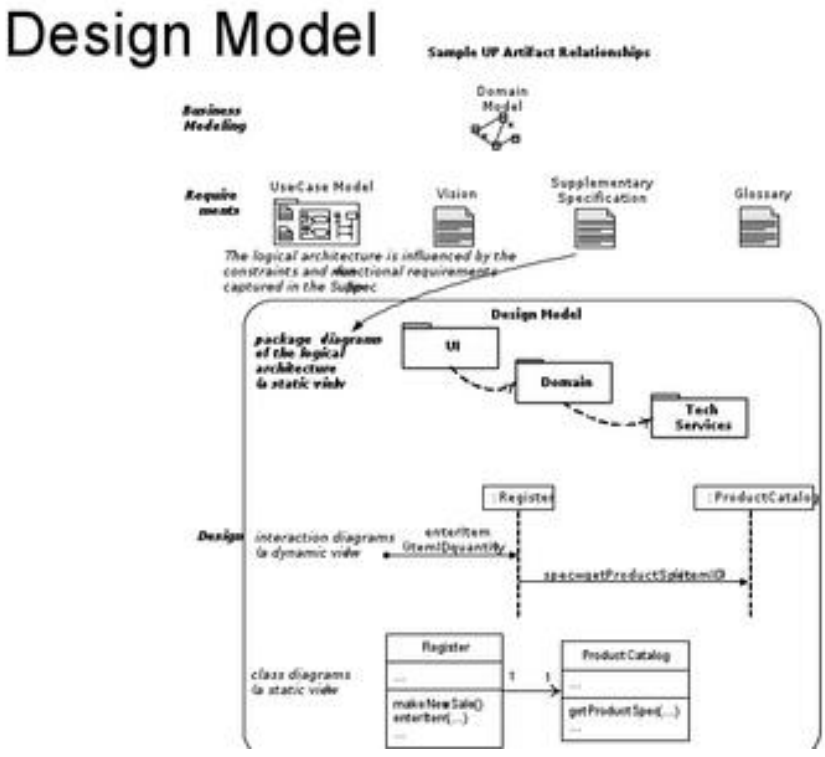


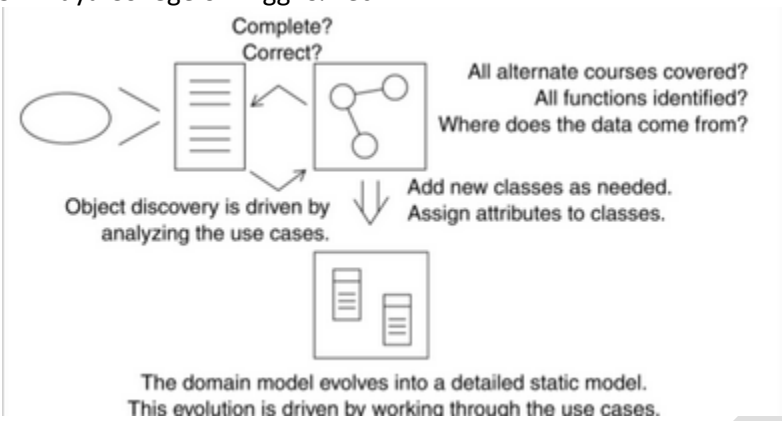


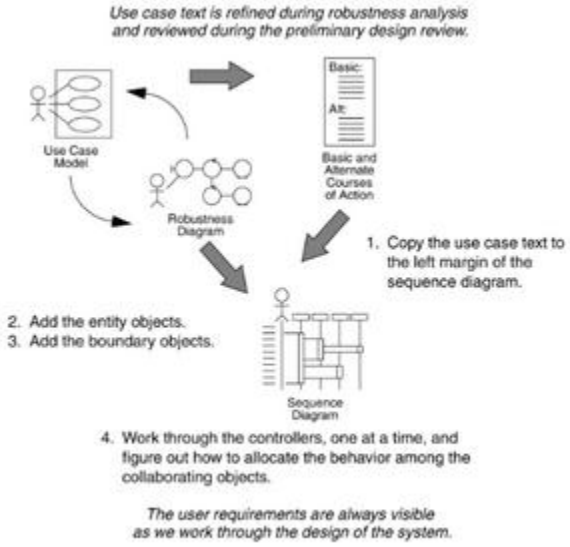
GRASP Principles

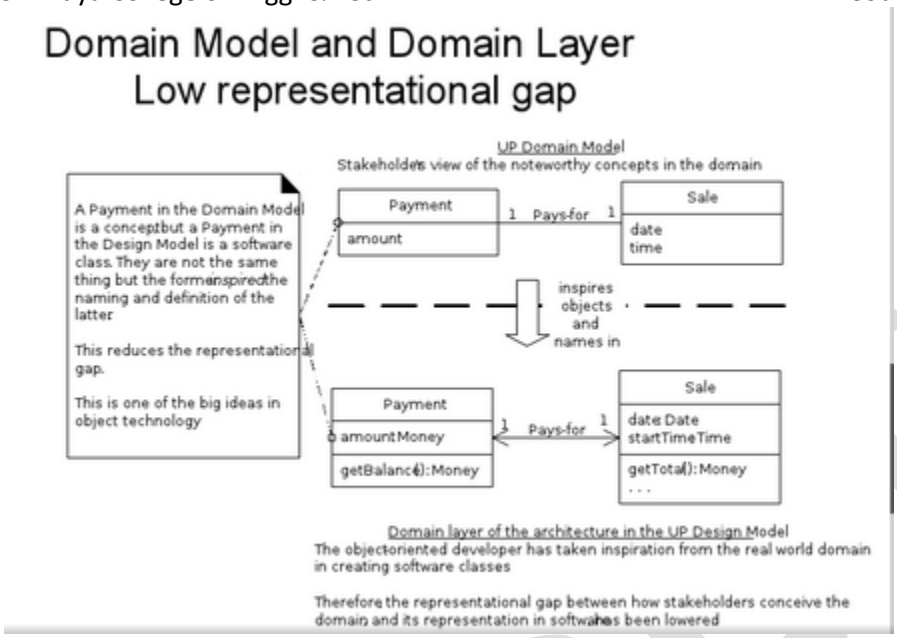
Design and use case realization

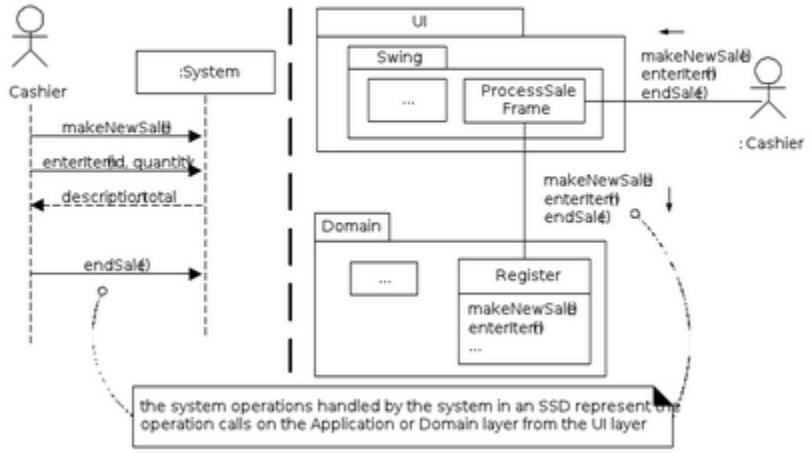
Design patterns




Complete?
Correct?
All alternate courses covered?
All functions identified?
Where does the data come from?
Object discovery is driven by analyzing the use cases.
Add new classes as needed.
Assign attributes to classes.
The domain model evolves into a detailed static model.
This evolution is driven by working through the use cases.


Use case text is refined during robustness analysis and reviewed during the preliminary design review.
Use Case Model
Basic:
Alt:
Basic and Alternate Courses of Action
Robustness Diagram
1. Copy the use case text to the left margin of the sequence diagram.
2. Add the entity objects.
3. Add the boundary objects.
Sequence Diagram
4. Work through the controllers, one at a time, and figure out how to allocate the behavior among the collaborating objects.
The user requirements are always visible as we work through the design of the system.





Designing for Visibility

Fact: To send a message to B, A must have visibility to B. It doesn't happen by "magic."

Kinds of visibility:

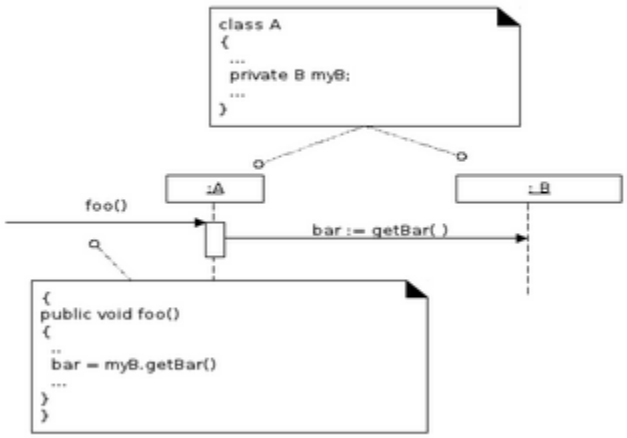
Attribute

Parameter

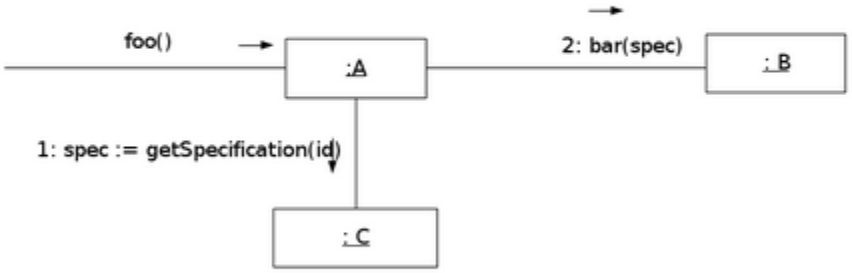
Local

Global

Attribute Visibility



Parameter Visibility



Local Visibility