(11224) INTRODUCTION TO SOFTWARE ENGINEERING

Lecture 10: Design Issues and Use Cases

Shereen Fouad

Good Design Heuristics

Separation of Concerns

- separate a system into different parts that overlap as little as possible.
- If you can find a way to split up a program into separate units, each of which deals with one aspect of the system and has very little overlap with other parts, then when you design or implement that one part, you do not need to complicate your work with consideration of details of other parts.
- concerns that are separated are usually pieces of functionality.

This might represent **higher cohesion**

```
class MyReader{
 public MyData readFromDisk(Parameter fileName){...};
 public MyData readFromWeb(Parameter url){...};
 public MyData readFromNetwork(Parameter networkLoc){...};
}
```

Look how well-focused the class is in its purpose. The class is named "MyReader" and its intended purpose is to read the resource, and it does only so. It does not implement other things. Its highly cohesive. Lets look at low cohesion example.

```
/*
 * lower cohesion example (probably)
 */
class MyReader{
 //validate the resource path
 public boolean validateLocation(String path){
  return ping(pathIP) && checkFTP(path);
 }
 private static boolean ping(String path){...};
 private static boolean checkFTP(String path){...};

 //read the resource
 public MyData readFromDisk(String fileName){...};
 public MyData readFromWeb(String url){...};
 public MyData readFromNetwork(String networkAddress){...};
}
```

Well, the read operations are well defined, but it also implements logic to validate path. This is just an example and describes lower cohesion, wherein, a class intended for a specific functionality is also involved in other operations.

http://javabambino.blogspot.co.uk/2013/06/loose-coupling-and-high-cohesion-in-java.html

Encapsulation

- Encapsulation are a way of directly encoding separation of concerns
- This means to wrap functionality, both data and behaviour, into a single unit, usually a module but also individual classes.
- This keeps that functionality together and promotes high cohesion.

Information Hiding

- One way the help ensure good modularity with loose coupling is to use information hiding.
- It means to control what parts of a unit (either a module or a class) is visible to other units.
- The basic idea is that if code chunk A doesn't really need to know something about how code chunk B (which it calls) does its job, don't make it know it. Then, when that part of B changes, you don't have to go back and change A.

It is not the same as encapsulation

- Encapsulation means collecting a bunch of stuff together and putting it in one box, or capsule.
- The box may or may not have opaque walls, so this may or may not involve information hiding.
- one can encapsulate functionality into, for example, a Java class, but leave the instance variables on public access for the world to see and manipulate.
- The functionality is encapsulated, but the information in the class is not hidden.

In Terms of Java

- There are many techniques and tricks to improve information hiding in Java, starting with making instance variables private.
- Other techniques such as making default constructors private, making classes final etc.

Quiz !!

A design is said to be a good design if the components are:

a) Strongly coupled
b) Weakly cohesive
c) Strongly coupled and Weakly cohesive
d) Strongly coupled and strongly cohesive
e) Strongly cohesive and weakly coupled

- **Which is called the specification of the number of occurrences of one object that can be related to the number of occurrences of another object?**

a) Encapsulation.
b) Relationship.
c) Modality.
d) Cardinality.

Quiz !!

- **Cohesion which occurs when a module performs one and only one computation and then returns a result is called _____.**

a) Layer Cohesion.
b) Functional Cohesion.
c) Communication Cohesion.
d) Sequential Cohesion.

- **When the model is analyzed, try to minimize _____.**

a) Cohesion.
b) Coupling.
c) Functions.
d) Complexity.

Use Cases

Use Cases

- A use case is a description of one operation on a system.
- It describes the requirements of a system in terms of the interactions that should occur between the system and various people and other systems that can interact with it.
- Use cases can be represented as textual documents or in a diagram
- Today we will cover the textual documents only

```
Name          Add Book to Basket
Initiator     User
Goal          Add a book to a customer's basket
Main Success Scenario
1.    User requests that a book be added to a customer's basket
2.    System asks User for customer's email and book's isbn
3.    User submits customer's email and book's isbn
4.    System identifies customer from email address
5.    System identifies book from ISBN
6.    System obtains current basket for customer
7.    System adds new item detail for book to basket

Extension
3.    User cancelled operation
      1. Fail
4.    No matching customer found
      1. System notifies User that Customer email address is invalid
      2. Resume 2
5.    No matching book found
      1. Include use case ''Search for Book''
5.1.  Use case ''Search for Book'' failed
      1. Fail
6.    No basket for customer found
      1. System adds basket for customer
7.    Item detail for this book exists in basket
      1. System adds one to quantity for item detail
      2. Stop
```

Here is an example of a use case that one might find in the specification of an online bookshop

Benefits of Use Cases

- primary vehicle for requirements capture
- described using the language of the customer (language of the domain which is defined in the glossary)
- provide an easily-understood communication mechanism
- When requirements are traced, they make it difficult for requirements to fall through the cracks
- provide a concise summary of what the system should do at an abstract (low modification cost) level.

Use Case Rules

Each use case must have

1. a unique name.
2. an initiator: this is an actor who starts the execution of a use case.
3. a goal: a short description of the purpose of the use case.
4. a single numbered sequence of steps that describe the main success scenario.

The following rules apply to the steps:

4.1 Each step must be of the form "**Actor does something**"

4.2 The first step indicates the **stimulus** (i.e., the event) that caused the use case to be initiated.

- The combination of Actor and Stimulus must be unique across the full set of all use cases.

4.3 Steps cannot use control flow operations such as parallelism, recursion, iteration, gotos or conditional statements.

- Thus no IF, REPEAT, or LOOP statements are allowed.

4.4 A step can invoke another use case (though not recursively) by naming it in an Include statement.

- In such a case, when the step is executed, the included use case is executed and, when it has terminated, execution continues with the next step in the first use case.

5. A use case may have a list of extensions

- Each extension is a mini-use case (including extension steps) that **describes an alternative** or addition to the main success scenario.
- They are described after the main success scenario
- An extension is made up of:
    1. The step number in the main scenario or extension to which this extension applies.
    2. A condition that triggers the extension if it becomes true on the indicated step. The condition is interpreted as follows:
        - If the condition evaluates to true, the extension is executed.
        - If it evaluates to false and there are more extensions defined for the same step, then execution continues with the next sequential extension for that step.
        - If it evaluates to false and there are no more extensions defined for the same step, then execution continues at the next step.

3. A numbered sequence of steps that constitute the extension

- Fail: the use case is terminated with the goal unsatisfied.
- Stop: the use case is terminated with the goal satisfied.
- Resume N: the extension ends and the use case continues by executing step N and continuing from there.
- None of the above: the use case continues restarting on the step that the extension applied to.

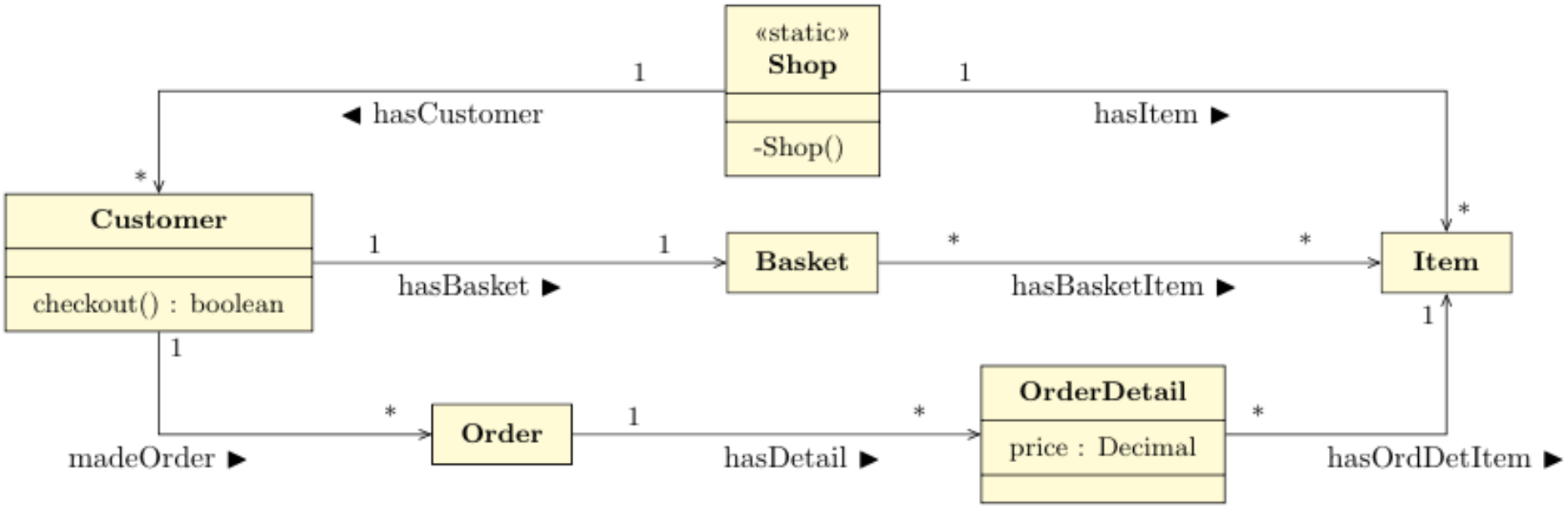
Name Add Book to Basket
Initiator User
Goal Add a book to a customer's basket
Main Success Scenario
1. User requests that a book be added to a customer's basket
2. System asks User for customer's email and book's isbn
3. User submits customer's email and book's isbn
4. System identifies customer from email address
5. System identifies book from ISBN
6. System obtains current basket for customer
7. System adds new item detail for book to basket

Extension
3. User cancelled operation
    1. Fail
4. No matching customer found
    1. System notifies User that Customer email address is invalid
    2. Resume 2
5. No matching book found
    1. Include use case ''Search for Book''
5.1. Use case ''Search for Book'' failed
    1. Fail
6. No basket for customer found
    1. System adds basket for customer
7. Item detail for this book exists in basket
    1. System adds one to quantity for item detail
    2. Stop

Shopping basket example

- A client wishes to build an online shopping system that will allow her to compete with Amazon in selling books. In the future, the system should be expandable so that items other than books can be sold (i.e. CDs, DVDs, etc.). As an initial proof of concept, the client wants a basic system that allows customers to log in, select items from a catalog and add them to a virtual basket of items. When the Customer is satisfied with their selection, they can choose to check out and buy the items in the basket. Customer's baskets should be maintained even if they log out without checking out, and the next time they log in, they can continue to add (or remove) items to (from) their basket. The price of items is kept in the catalog and may change from day to day. Therefore the price of items in a customers basket may change from day to day. However, when a customer checks out, the price of the items in the order is fixed at that time for this order, even if it takes some time to source the items for delivery and the item price in the catalog has changed during the delay. All details of old orders should be kept so that the Customer can review old orders.

Shopping basket example



```
Name         Checkout
Initiator    Customer
Goal         Collect all items currently in customer's basket, take
             payment and schedule order fulfilment.

Main Success Scenario
1.   Customer requests to checkout
2.   System obtains current basket for customer
3.   System creates new order for customer
4.   System adds order details from basket to order
5.   Include Use Case "Finalise Order and Collect Payment"
6.   System dispatches order to OrderFulfilment
```

Extensions ??

```
Name         Checkout
Initiator    Customer
Goal         Collect all items currently in customer's basket, take
             payment and schedule order fulfilment.

Main Success Scenario
1.   Customer requests to checkout
2.   System obtains current basket for customer
3.   System creates new order for customer
4.   System adds order details from basket to order
5.   Include Use Case "Finalise Order and Collect Payment"
6.   System dispatches order to OrderFulfilment

Extension
3.   No basket for customer found
     1. Fail
3.   Basket for customer is empty
     1. Fail
6.   Payment collection failed
     1. Fail
```

UML Diagrams

Structural

: element of spec. irrespective of time

- Class
- Component
- Deployment
- Object
- Composite structure
- Package

Behavioral

: behavioral features of a system / business process

- Activity
- State machine
- **Use case**
- Interaction

Use cases have become a diagram type in the Unified Modelling Language (UML)