Universitas Esa Unggul

Smart, Creative and Entrepreneurial

CPL230-PENGEMBANGAN PERANGKAT LUNAK (PERTEMUAN-7)

www.esaunggul.ac.id

Dosen Pengampu :

5165-Kundang K Juman

Prodi Teknik Informatika Fakultas Ilmu Komputer

Recall The Team Skills

1. Analyzing the Problem (with 5 steps)
2. Understanding User and Stakeholder Needs
3. Defining the System
4. Managing Scope
5. **Refining the System Definition**
    1. Software Requirements: a more rigorous look
    2. Refining the Use cases
    3. Developing the Supplementary Specification
    4. On Ambiguity and Specificity
    5. Technical Methods for Specifying Requirements
6. Building the Right System

Chapter 21

Refining the Use Cases

- How Use Cases Evolve
- The Scope of a Use Case
- Dependency Relationships
- Extending Use Cases
- Including Use Cases

How Use Cases Evolve

- The test for enough use cases should be the following:
- A complete collection of use cases should describe
  - all possible ways in which the system can be used,
  - at a level of specificity suitable to drive design, implementation, and testing.

The Scope of a Use Case

- Consider the use of a recycling machine.
- The customer
  - inserts cans and bottles into the recycling machine,
  - presses a button,
  - and receives a printed receipt that can be exchanged for money.
- Are there 3 uses cases?
  - one use case to insert a deposit item,
  - another use case to press the button,
  - and another to acquire the receipt?
- Or is it just one use case?

The Scope of a Use Case

- Three actions occur, but one without the others is of little value to the customer.
- The complete process is required to make sense to the customer.
- Thus, the complete dialogue
  - from inserting the first deposit item to pressing the button to getting the receipt

is a complete instance of use, of one use case.

Review use case

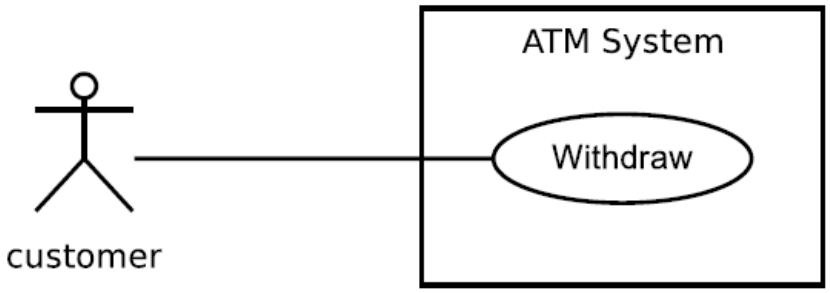
- Review name
  - Turn light on/off
  - Control light
- Refining the description
  - The resident initiates a change to the light the room by pressing the on/off switch in the room-lighting control panel.
  - The use case prescribes the way in which lights are turned on and off and also how they are dimmed and brightened in accordance with how long the user presses a light switch

Pre/Post condition

- Pre condition
  - State of the system
  - That the user can observe (observable state), not the event that starts the use case
    - The user has logged on to the system
    - The user has opened the doc
- Post condition
  - describes what the change in state of the system will be after the use case completes.
  - Post-conditions are guaranteed to be true when the use case ends
    - a cash withdraw will lead to an update of the account

Example: Automated Teller Machine

In this case study, look only at withdrawal:



Main flow:

insert card, enter PIN, request cash amount, get money, return card

Alternative flows: wrong PIN, cash amount exceeds balance, etc.

In this work: put a more precise informal description in the *post condition* of the use case. . .

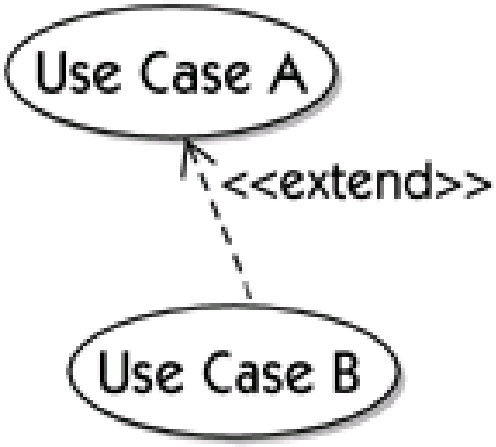
Post Condition of Withdraw

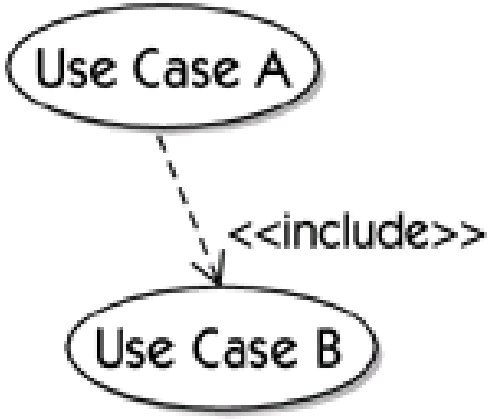
- If the customer entered the PIN on the Card, and the customer's balance was greater or equal to the requested amount, then the customer got the requested amount and the amount was deducted from the balance.
- If the customer entered the wrong PIN three times, the card was retained.
- If the customer requested too much money, the card was returned to the customer.

- Control light use case
  - Pre-conditions
    - The selected On/Off/Dim button must be Dim Enabled
    - The selected On/Off/Dim button must be preprogrammed to control a Light Bank
  - Post-conditions
    - On leaving this use case, the system remembers the current brightness level of the selected On/Off/Dim button.

Dependency Relationships between Use Cases

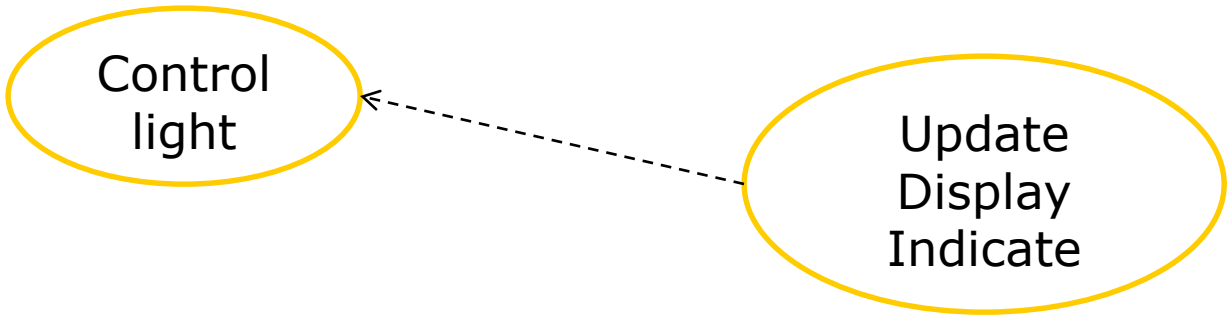
- Extend relationship defines that instances of a use case that may be augmented by some additional behaviour in an extended use case.
- Include relationship is a directed relationship between use cases, implying that the behaviour in the additional use case is inserted into the behaviour of the base use case.





Extending Use Cases

- Systems evolve over time and additional features and functionality are added.
- A use case may be extended to have more actions in certain conditions.
  - If some HOLIS systems included an optional "light bar" indicator on the control switch.

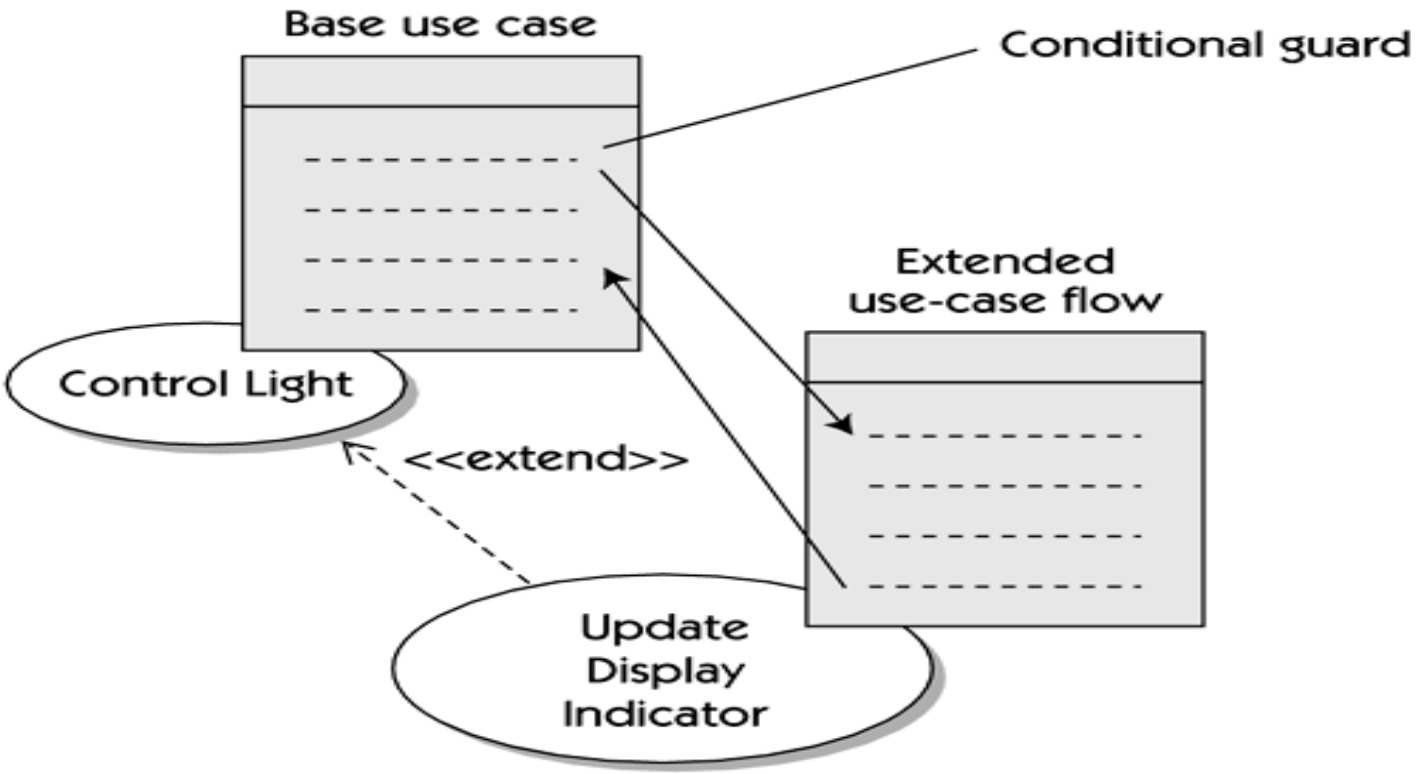


Extending Use Cases

- Why use the extend concept at all?

1. It can simplify maintenance and allow us to focus only on the extended functionality
2. Extension points for envisioned extensions can be provided in the base use case, which is an indication to future intent
3. The extended use case may represent optional behavior as opposed to a new, basic or alternative flow

A Base Use Case with Extended Flow

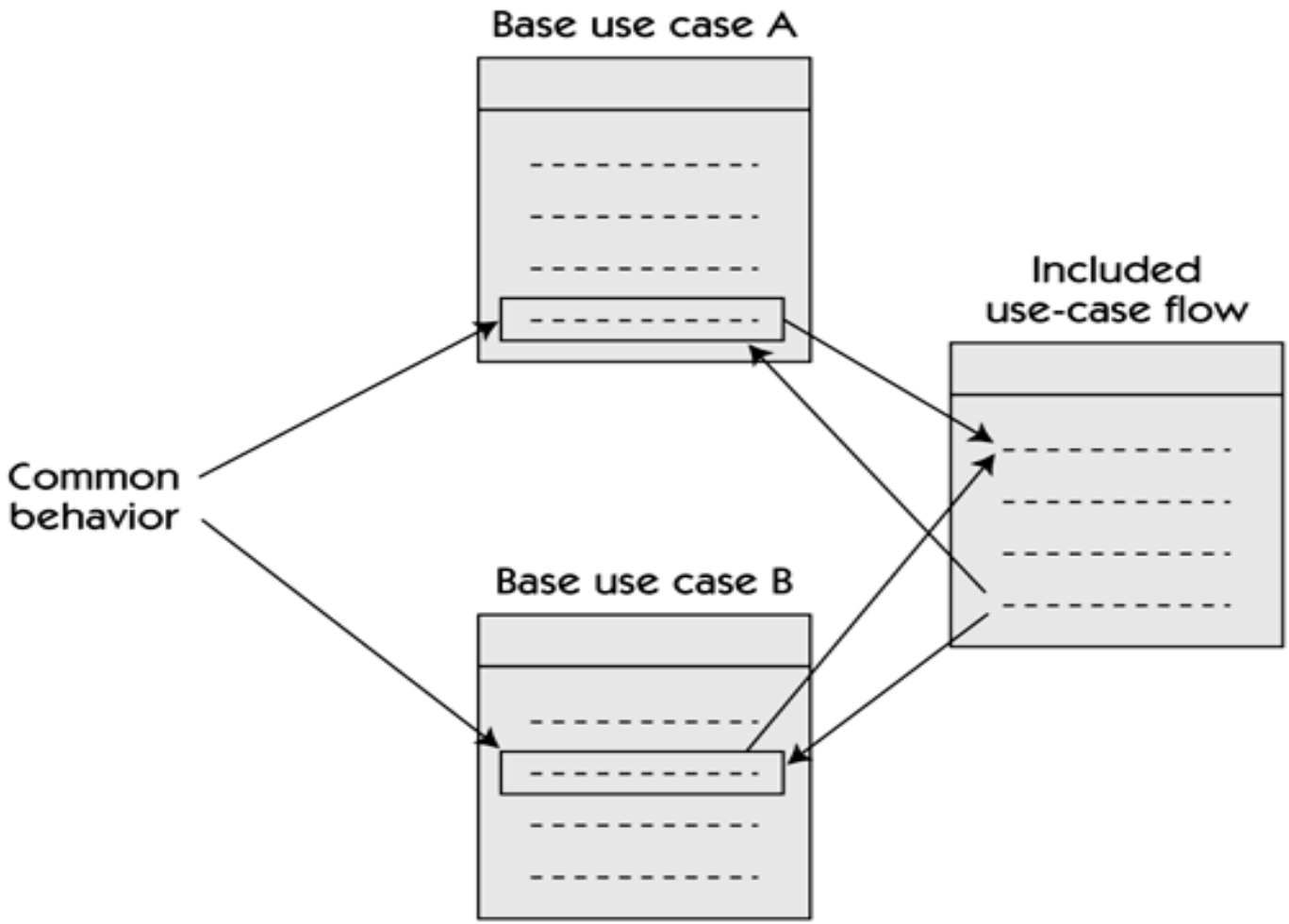


- In order to apply the extend construct, all that is required is to indicate the extension points in the basic flow and the conditions under which the extended flow is to be executed.

Including Use Cases in Other Use Cases

- Certain patterns of user and system behavior reoccur in a variety of places
- e.g., entering passwords, performing a system status check, selecting items from a table, etc.
- To avoid redundancy, the include relationship can be used.
- When used properly, the include relationship can simplify the development and maintenance activities.

The Flow of an Included Use Case



Key Points

- To support development and testing activities, the use cases defined earlier in the project must be more fully elaborated.
- The use-case model is reviewed and will often be refactored as well.
- A well-elaborated use case also defines all alternative flows, pre- and post-conditions, and special requirements.
- The additional use-case relationships extend and include help the team structure and maintain the use-case model.

Reading Assignment

- Read HOLIS case study in pages 245-251.