SCC.211 Software Design

Topic: Use Cases

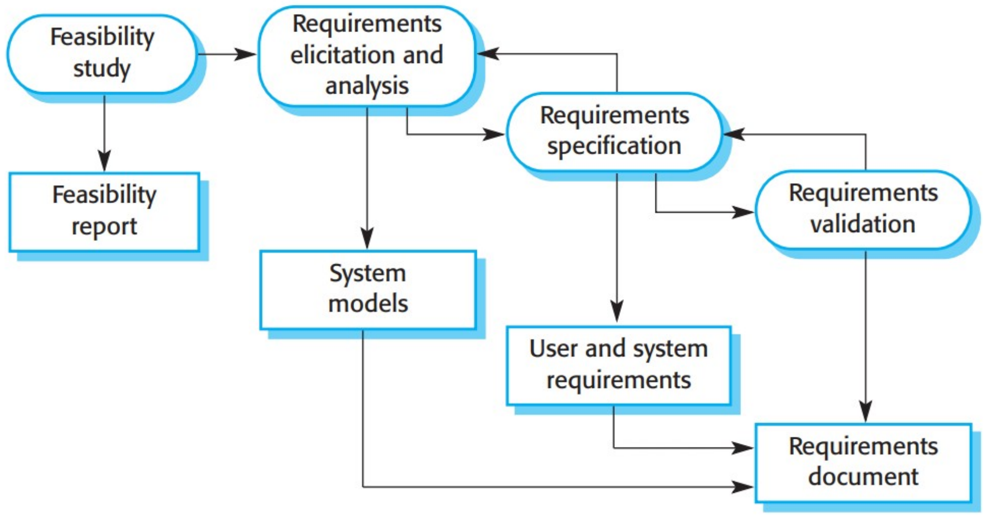
School of Computing and Communications
InfoLab21

Recap …

- In this course: ***systems engineering perspective*** on software design, starting with requirements.
  - Requirements (definition)

    [*Descriptions of **what** the system should do (functional requirements), along with any **constraints** under which it must do it (non-functional requirements).*]
  - Requirements engineering

    [*A systematic process of **understanding the problem** and specifying a solution*.]



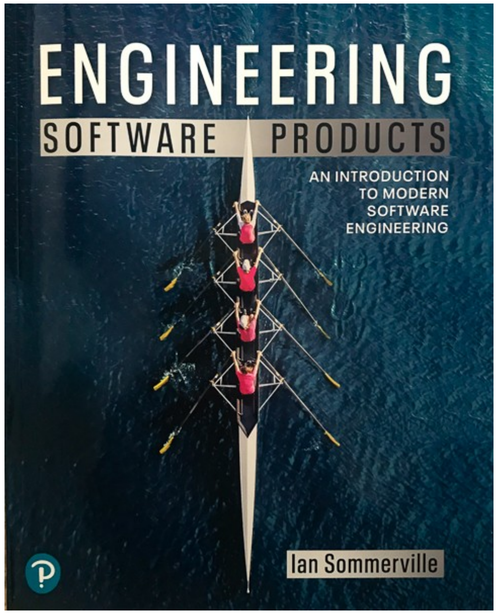
Other Perspectives on SWE

More *lightweight* (agile) development processes exist.

[! ***NOT PART OF THIS COURSE*** !]

Ian Sommerville : ***Engineering Software Products: an introduction to modern software engineering***, *2019.*

https://lancaster.primo.exlibrisgroup.com/permalink/44LAN_INST/1h0hp1j/alma9930665909901221



High Integrity Software

High-integrity systems can require the use of ***formal methods*** – a more rigorous (mathematically precise) approach than in SCC.211.

[! ***FORMAL METHODS NO LONGER PART OF THIS COURSE*** !]

– Neil Evans, ***Introduction to Formal Methods***, The Science & Technology Journal of AWE, Issue 22, October 2011; pp. 18-25. (https://www.awe.co.uk/wp-content/uploads/2014/09/68e180bAWE_Discovery_22-6.pdf)

– Formal Methods Specification and Analysis Guidebook for the Verification of Software and Computer Systems: A Practitioner's Companion - Volume II (https://ntrs.nasa.gov/api/citations/19980227975/downloads/19980227975.pdf) [*Only for those interested in formal methods: see pp. 1-25.*]

Requirements discovery (elicitation)

- How can we discover requirements?
  - From our understanding of the problem
  - So, we can specify the solution
- There are many ways to do this. The obvious one, asking users what they want.
  - But stakeholders *"don't always know what they really want"?*
  - Stakeholders express requirements in their own terms.
  - Different stakeholders may have conflicting needs.
  - The requirements change during the analysis process.
- The system developer needs to work with stakeholders, to understand their needs,
  - Use case analysis is one way to do this.

Could you please go to the store for me and buy one carton of milk, and if they have eggs, get 6!
30 MINUTES LATER...
Here you go!
MILK
Why the hell did you buy 6 bottles of milk ?
They had eggs!!
AnAr

When it goes wrong... (?)

"Some Navy aircraft were ferrying missiles from one point to another. One *pilot executed a planned test by aiming at the aircraft in front (as he had been told to do) and firing a dummy missile*. Apparently, nobody knew that the "smart" software was designed to substitute a different missile if the one that was commanded to be fired was not in a good position. In this case, there was an antenna between the dummy missile and the target, *so the software decided to fire a live missile located in a different (better) position instead*. What aircraft component(s) failed here? "

Nancy Leveson, Are You Sure Your Software Will Not Kill Anyone? Comm. ACM, 2020.

Use cases[1]

- Offer a way to structure requirements according to stakeholder needs/goals
- They are useful because:
  - They describe “scenarios” or sample execution scenarios for the system under development, which are easy to understand.
  - They are **mainly** textual so easy to use as a way to communicate between different stakeholders.
  - They can be used to derive test cases.
  - They are a standard part of the **U**nified **M**odelling **L**anguage

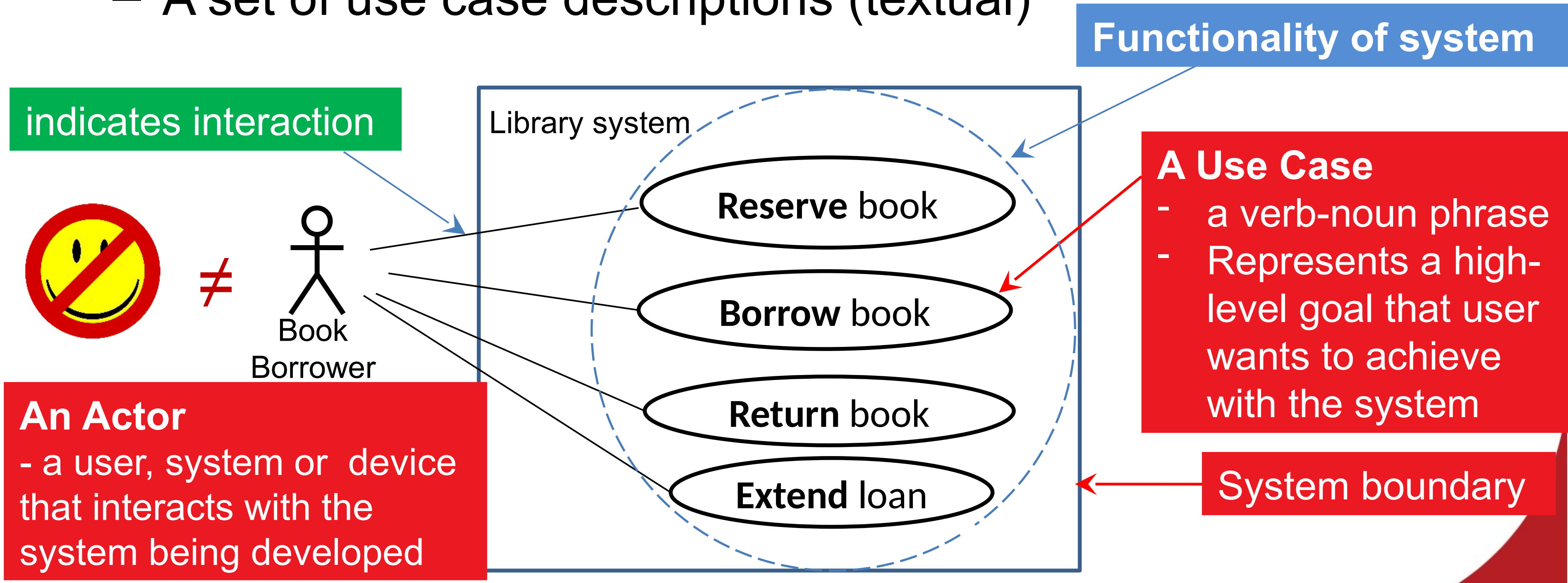
1. **Jacobson Ivar,** Object-oriented software development in an industrial environment. In Conference Proceedings of *Object-oriented Programming Systems, Languages, and Applications* (OOPSLA 87), pp. 183-191, Dec1987.

Before we go on… lets talk a bit about modelling

- Models are use extensively in software engineering – informal graphical (diagrams), semi-formal, formal – pls look up *formal methods*)
  - A simplification of reality
  - Why do we use models in SE?
    - *To deal with complexity*: abstraction, separation of concerns
      - For example, **UML** (Unified Modeling Language) allows you describe several system models,
        - » Class, object, & component models etc., model static relationships
        - » Statecharts, activity diagrams, sequence diagrams model dynamic relationships
  - Are they useful?
    - Yes – they allows us to focus on specific concerns and hide unnecessary detail
  - We can perform consistency checking among different models

Use cases

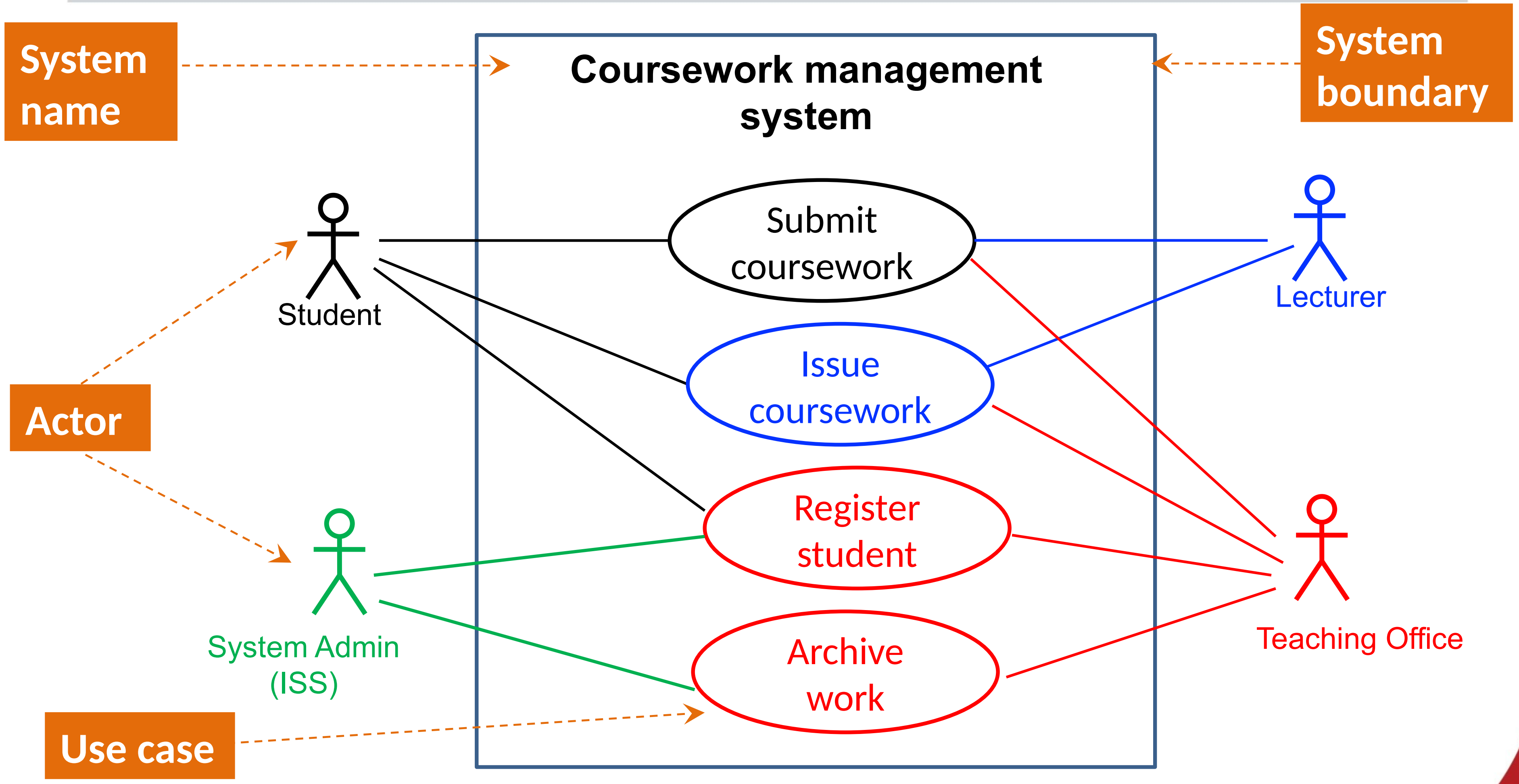
- A use case is a **behavioural model** of a system.
  - It describes how a system responds to stimuli from "outside" of the system.
- A use case model consists of:
  - A use case diagram (see below)
  - A set of use case descriptions (textual)



Use case descriptions

- The diagrams serve as a high-level reference. They tell us:
  - What use cases are there for the system under development?
  - Which actors interact with each use case?
- Use cases need to be developed to find out:
  - What happens <u>and in what order</u> – i.e., how the use case responds to a stimulus
  - What the actors require/need.
- <u>Use case descriptions</u> are used to develop use cases
  - Use case descriptions are developed as **scenarios**

Use case diagram for a coursework management system



Use case description: **Submit coursework** (1)

Use case template

Scope: System name

Primary actor:

Secondary actors:

Preconditions

Activities that must take place, or any condition that must be true, before the use case can be started

Flow of events

User actions and system responses that will take place during execution of the use case under normal conditions

<May include alternate flows> An **alternate flow** describes a scenario other than the basic **flow** that results in a user **completing or achieving** their goal.

Postconditions

State of the system at the conclusion of the execution with normal flow of events

Submit course use case description

Preconditions:

1. Student has been issued with the coursework
2. Student has attempted the coursework.

Flow of events:

<described in the next slide>

Postconditions:

1. Student receives mark.
2. Student's mark is recorded.

Use case description: Submit coursework (2)

- Flow of events:
  1. Student submits coursework.
  2. Teaching Office records submission.
  3. Teaching Office passes submission to tutor.
  4. Lecturer marks submission.
  5. Lecturer returns marked submission.
  6. Teaching Office records mark.
  7. Teaching Office returns mark to student.
- But … this is only one possible sequence of events for this use case:

What about late submissions?

- We could introduce control structures into our description
    1. Student submits coursework.
    2. Teaching Office records submission.
    3. **If deadline expired**
        - 3.1. Teaching Office **marks submission as ‘late’ with date**
    4. Teaching Office passes submission to tutor.
    5. ….
- This approach is too procedural/ too algorithmic:
    - Essential properties get obscured with detail.
    - It’s hard to know when to stop - we tend to write algorithmic solutions rather than capturing what is the essential behaviour

Instead, use scenarios

- They describe:
  - A **primary** (‘happy day’) scenario:
    - Captures what we expect to be the normal behaviour.
  - Some **secondary** scenarios:
    - That reveal useful information: e.g., exceptions
- Hence, a use case is a collection of revealing scenarios.
- The best way to develop the [user] scenarios is by working with the user(s).

Submit coursework (3)

- Flow of events:
  - Primary scenario
    - 1. Student submits coursework.
    - 2. Teaching Office records submission.
    - 3. Teaching Office passes submission to tutor.
    - 4. …
  - Secondary scenarios
    - ***Late submission sequence***

Submit coursework (4)

- **Late submission sequence**

  Flow of events:

  1. Coursework deadline expires.
  2. Student submits coursework.
  3. Teaching Office records submission.
  4. **Teaching Office flags submission as ‘late’ with date.**
  5. Teaching Office passes submission to Lecturer.
  6. Lecturer returns marked submission.
  7. **Teaching Office records mark**.
  8. **Teaching Office returns mark.**

Scenarios and requirements

- Use cases are often used to derive the type of <u>functional requirements</u> that we saw in Week 1.
  - The enable us to add detail and structure to the requirements
  - In some cases, use cases are used as requirement specifications.
- Because of what they reveal about actions by the user and the system, and about sequencing, events are the primary sources of the requirements.
- Use cases should be developed by the users (actors) in a workshop run by the developer/engineer/analyst.

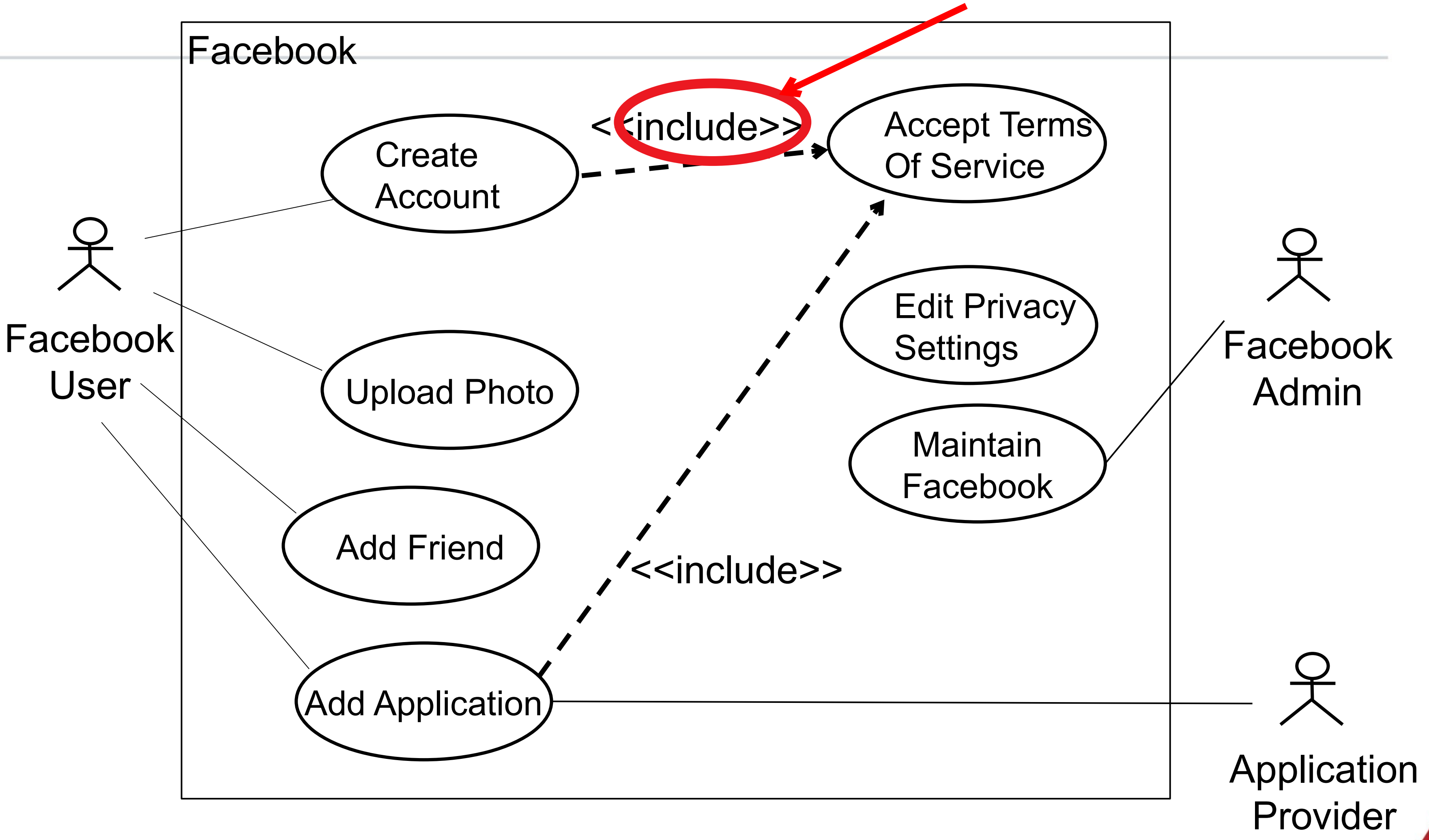
Submit coursework: Derived requirements

Note: each requirement is uniquely identified. This is needed to manage them.

- **CMS1.1** A student **shall** be able to submit a coursework assignment to the system once the lecturer has issued the assignment.
- **CMS1.2** On receipt of a coursework submission, the system **shall** record the assignment, the name of the submitting student and the time and date.
- **CMS1.3** On expiry of the coursework deadline the system **shall** issue a report to the lecturer and Teaching Office listing the names, and submissions dates and times of each student registered on the module.
- **CMS1.4** After the expiry of the coursework submission deadline, the system **shall** alert the lecturer and Teaching Office, and amend the record of the submitting student in the submission report to record the submission time and date and **shall** mark the record 'LATE'.
- **CMS1.5** The lecturer **shall** be able to access the coursework submissions received by the system after issuing the coursework assignment.
- **CMS1.6** …….

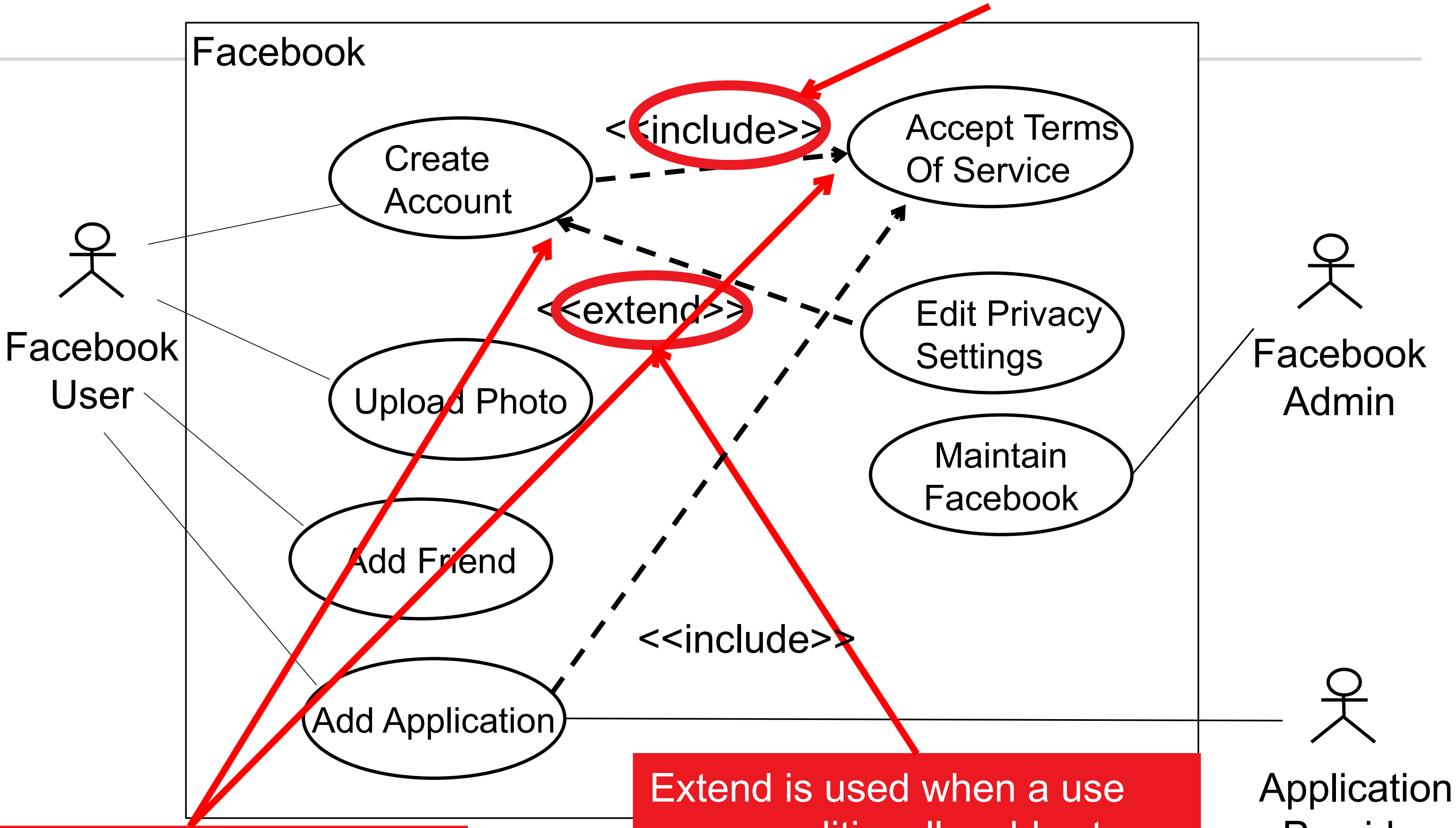
Use case: Facebook example

Include is used to extract use case fragments that are duplicated in multiple use cases (i.e., common mandatory behaviour)



Use case: Facebook example

Include is used to extract use case fragments that are duplicated in multiple use cases (i.e., common mandatory behaviour)



Extend is used when a use case conditionally adds steps to another use case
(i.e., optional behaviour)

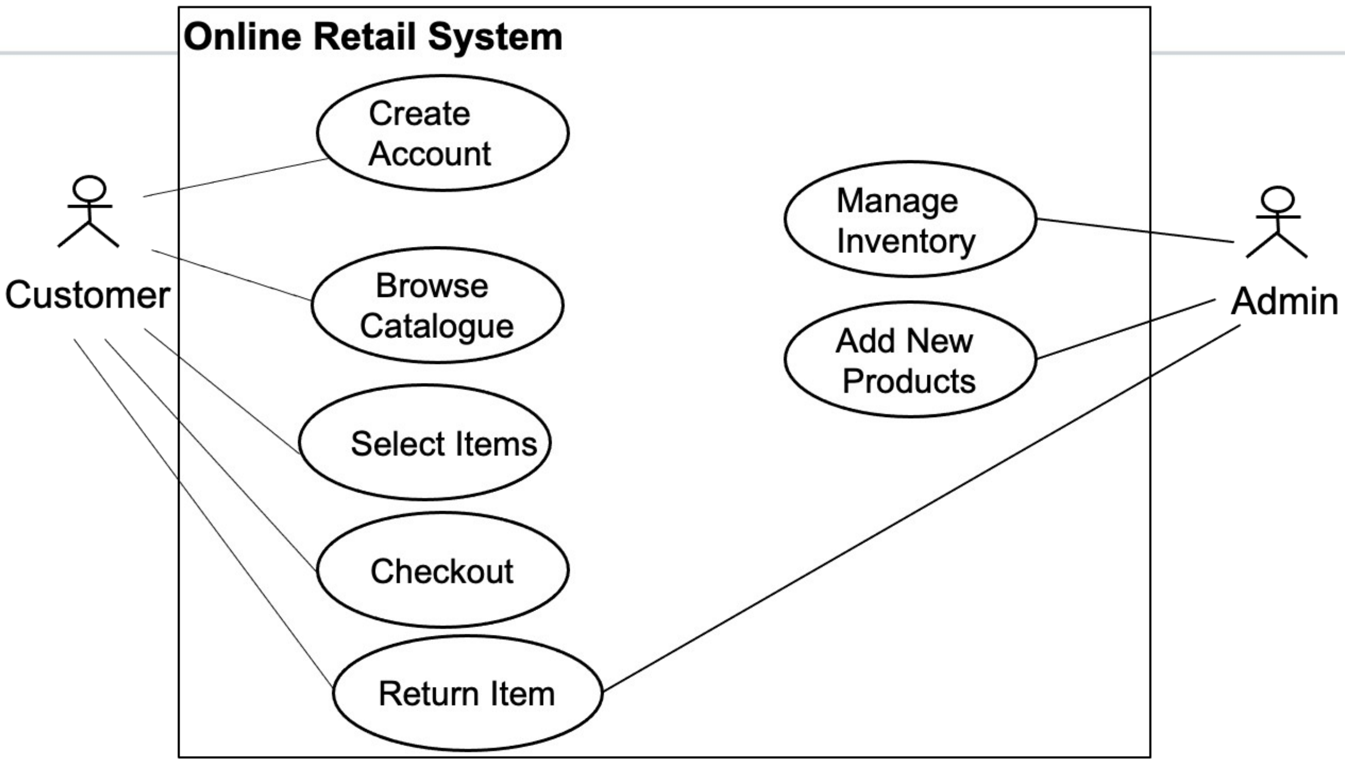
Be aware of the two different directions of the arrows!!!

Use case name:	***Create Account***
Scope:	Facebook
Primary actor:	*Facebook User.* <A Primary actor is an actor using the system to achieve a goal>
Secondary actors:	None. <A secondary actor is an actor that the system needs assistance from to achieve the primary actor's goal>
Summary:	*Facebook User* intends to create a new facebook account
Preconditions:	*Facebook User* is at the facebook website
Main success scenario:	*1.* *Facebook User* enters user details (name, email, password, sex, DOB) into *Facebook* and requests signup *2.* *Facebook* validates user details & presents terms of service *3.* *Facebook User* <u>Accepts Terms of Service</u> *4.* *Facebook* creates user account and asks *Facebook User* to create a profile *5.* *Facebook User* defers <u>Editing Privacy Settings</u> *6.* *Facebook* stores profile information 7. Facebook presents welcome page to *Facebook User*
Alternatives:	5a.1 *Facebook User* <u>Edits Privacy Settings</u>
Exceptions:	2a. User details cannot be validated. Registration is denied
Postconditions:	*Facebook User* has a registered facebook account
Non-functional reqs	<You may add non-functional requirements here>

Alternatives: different ways to fulfil the postconditions

Exceptions: **cannot** fulfil the postconditions.

Primary and secondary actors?

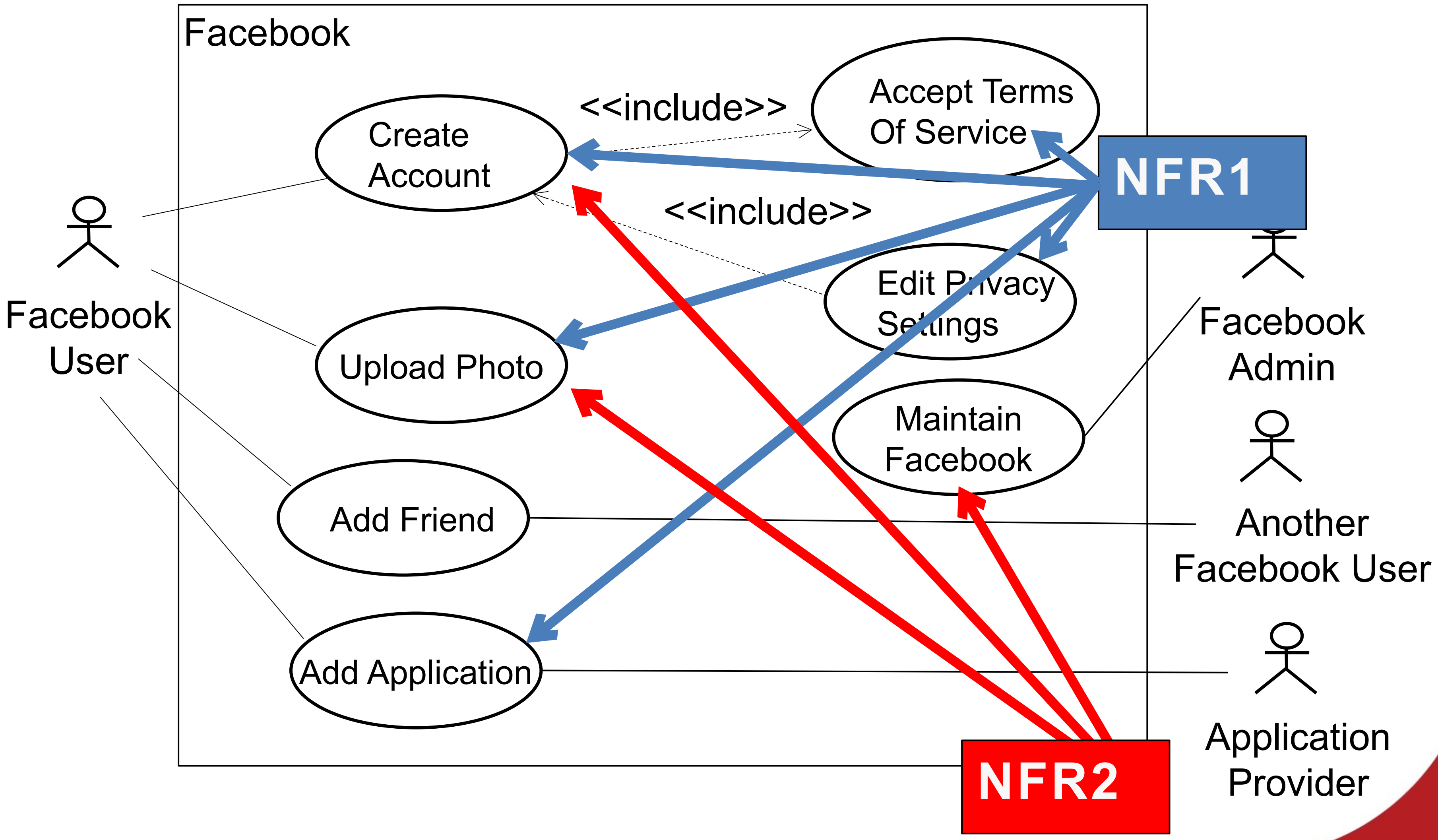


- **Customer** is Primary Actor, in the case Create Account, Browse Catalogue, Select Items, Return Item
  - **Admin** is Secondary Actor for Return Item
- **Admin** is Primary Actor in the case of Manage Inventory and Add New Products.

Non-functional requirements

- Do not appear (explicitly) in use cases
- However, simple annotations can be used to add them in most cases
- Facebook example.:
  - NFR1: All data stored by Facebook shall respect the privacy permissions set by the user
  - NFR2: Facebook should support 300 million active users with a growth rate of 1 million per month

Adding Non-functional requirements to use cases



Can you draw a use case diagram for this ATM example?

- A bank owns an automated teller machine (cashpoint) system that enables its customers to:
  - withdraw money from their account
  - display their account balance
  - display the last 10 transactions on their account
- Customers must be verified before they can access any of the ATM services
- Customer accounts are stored in the bank’s database

Can you draw a use case diagram for this ATM example?

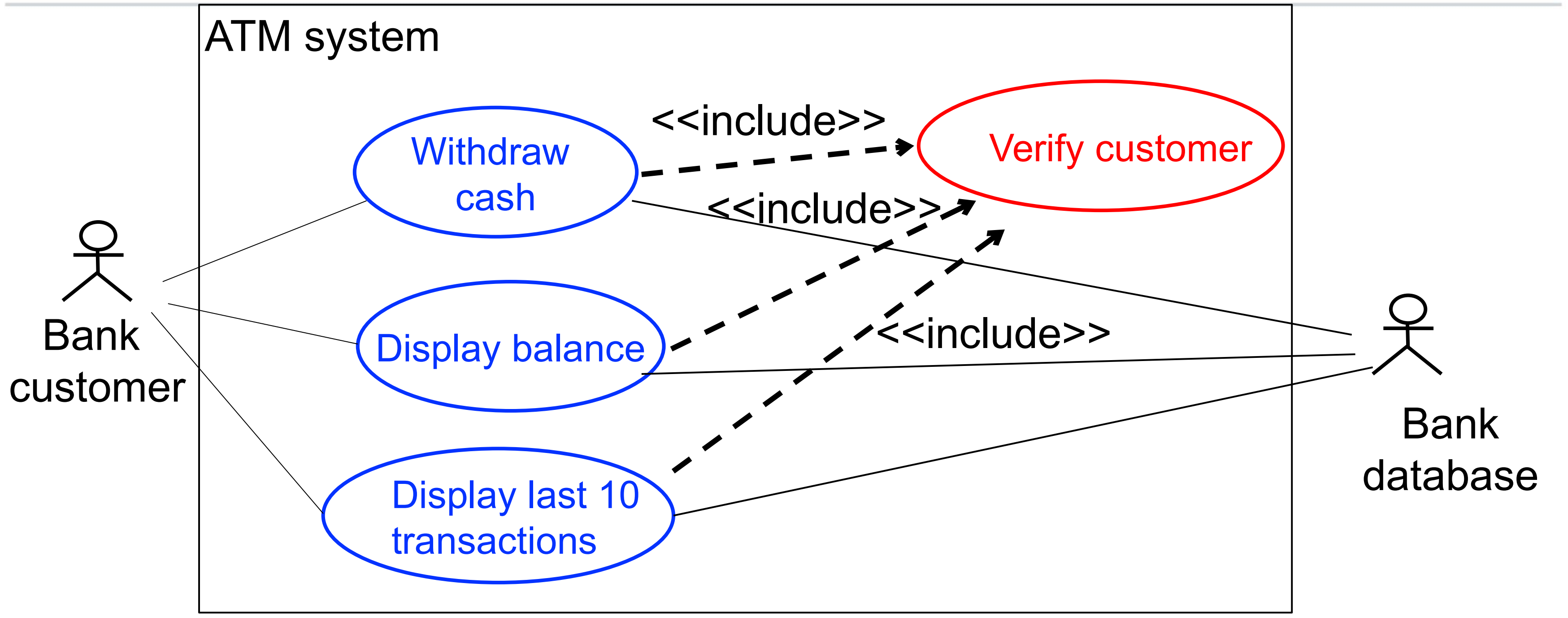
- A bank owns an automated teller machine (cashpoint) system that enables its **customers** to:
  - **withdraw money** from their account
  - **display** their **account balance**
  - **display the last 10 transactions** on their account
- Customers must be verified before they can access any of the ATM services
- Customer accounts are stored in the **bank's database**

Key

- Actor (RED)

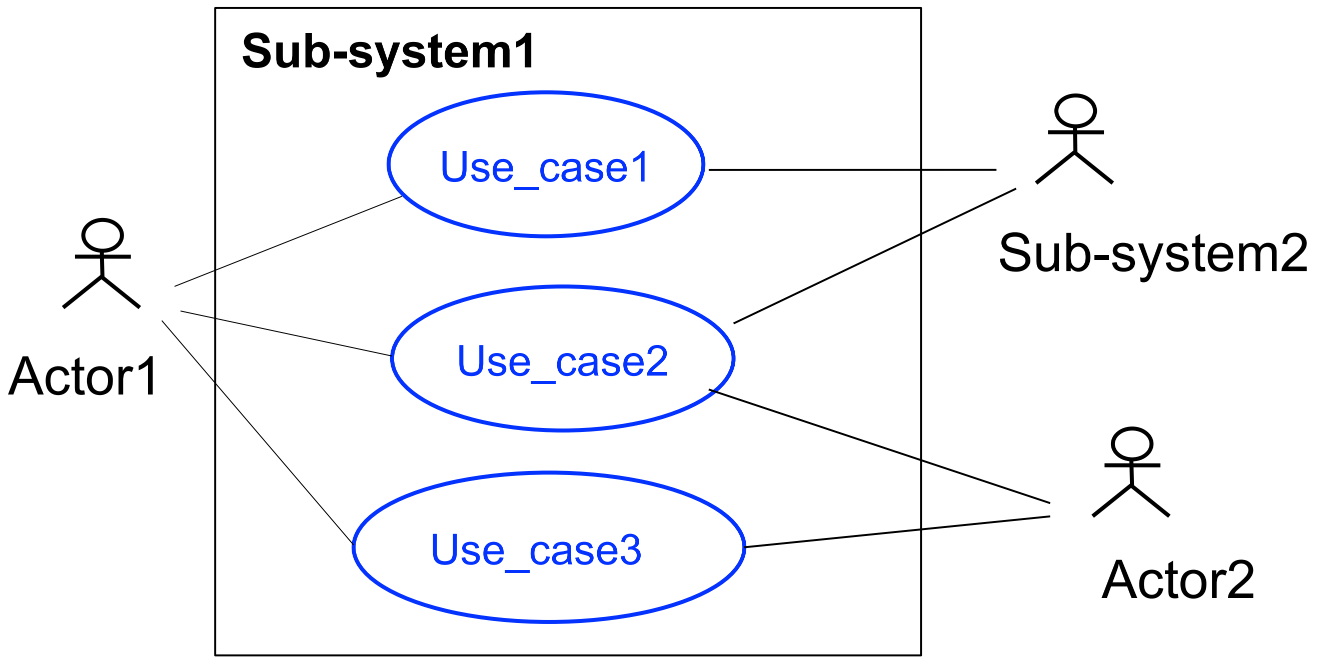
- Use Case (Blue) – Write as ***Action-Verb*** phrase

ATM example: Answer



- The primary actor is the bank customer whose intention is to achieve one or more of 3 the goals described represented by use cases (shown in blue)
- The customer account database is a secondary actor helping the customer to achieve his/her goal
- The 3 use cases all require the customer to be verified (see “red” use case)

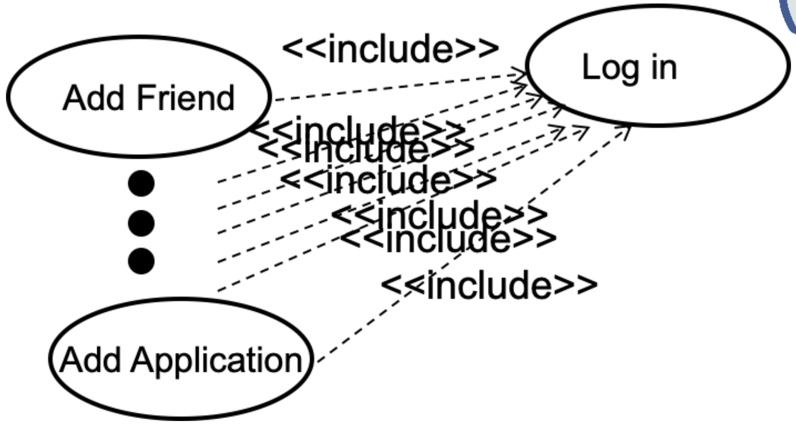
Managing use case diagram complexity

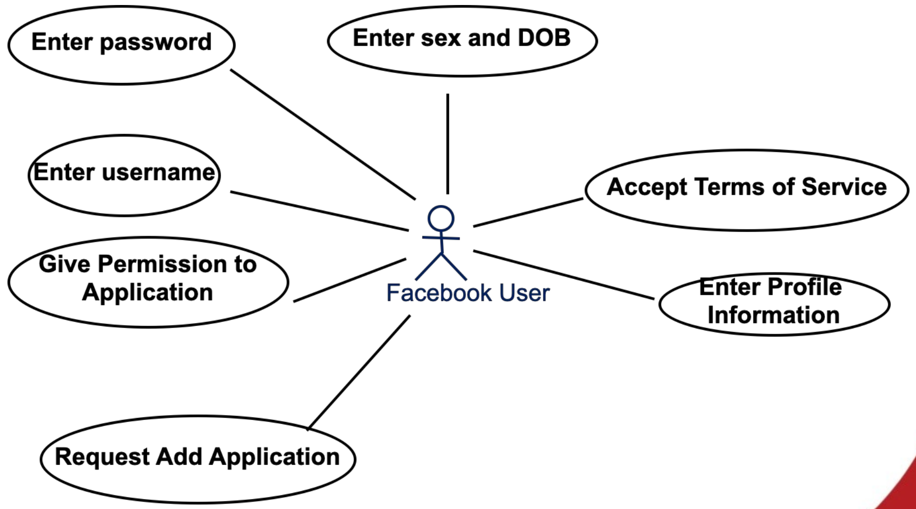


- **Sub-system1** captures all the use cases associated that sub-system
- **Sub-system2** interacts with **Sub-system1** and may be another use case diagram. However, as it is external to **Sub-system1**, it plays the role of an **actor**

Use case Guidelines

1. Avoid Excessive Structuring
2. Avoid Functional Decomposition
3. Understand the difference between **include** and **extend**
   - Only use when absolutely necessary





Use case guidelines: level of use case detail

Use cases can be written at different levels of detail

Cloud level. Very high level summary. Typically a business goal.

Sea level. Or user goal. Key characteristics: single sitting & user gets a result

Fish level. Usually the include or extend use cases

Bottom-of-the-ocean level. Too low!

Examples:

Cloud level. Sell Books Online

Sea level. Buy Book

Fish level. Login

Bottom-of-the-ocean level. Check password.

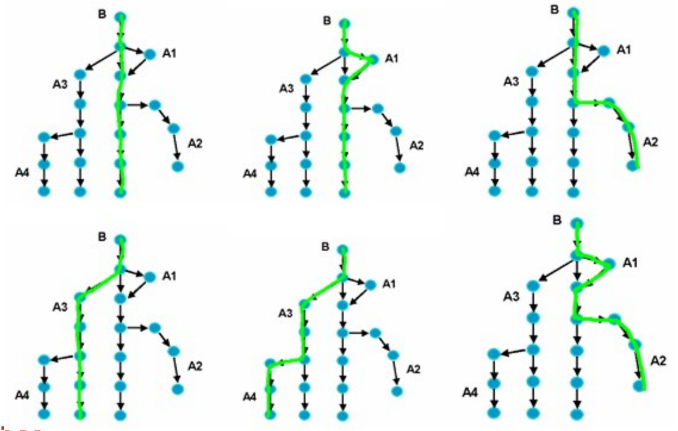
Facebook example: Different use levels

- Cloud:
  - Maximize number of users
  - Maintain Reputation
  - Secure User's Private Data
- Sea:
  - Add Application
  - Add Friend
  - Create Account
- Fish:
  - Login
  - Accept Terms of Service
- Bottom-of-Ocean
  - Enter Password
  - Click on "submit"

Use Case Testing

- Create (at least) one test case for each flow

- Identify variables for each use case step
  - Identify significantly different options for each variable
- Combine options to be tested into test cases
- Assign values for variables



For more details, see

1. https://www.softwaretestinghelp.com/use-case-te
sting/

Image from: http://www.ibm.com/developerworks/rational/library/04/r-3217/index.html?S_TACT=105AGX15&S_CMP=EDU

Further Reading on Use Cases

- Good exemplar of industrial-strength use case document:
  - http://www.cs.mcgill.ca/~joerg/taosd/TAOSD/TAOSD_files/AOM_Case_Study.pdf

Thank You!